Universidade Federal Fluminense

ANDRÉ CORDEIRO MACEDO MACIEL

Heurísticas para o Problema do Caixeiro Viajante Branco e Preto

NITERÓI

2005

ANDRÉ CORDEIRO MACEDO MACIEL

Heurísticas para o Problema do Caixeiro Viajante Branco e Preto

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização Combinatória.

Orientador:
Luiz Satoru Ochi

Co-orientador:
Carlos Alberto de J. Martinhon

Universidade Federal Fluminense

NITERÓI

2005

Heurísticas para o Problema do Caixeiro Viajante Branco e Preto

André Cordeiro Macedo Maciel

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

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

Prof. Carlos Alberto de J. Martinhon / IC-UFF
(Co-orientador)

Prof. Carlile Campos Lavor / IMECC-UNICAMP

Prof. Eduardo Uchoa Barboza / EP-UFF

Prof. José Elias Claudio Arroyo / UCAM-CAMPOS

Niterói, 26 de Agosto de 2005.

A civilização avança estendendo o número de operações importantes que podem ser realizadas sem que seja necessário pensar sobre elas.

Alfred North Whitehead

Agradecimentos

Uma dissertação de mestrado nunca é um projeto individual e ao longo do seu desenvolvimento foram várias as pessoas que, de uma forma ou de outra, contribuíram para que esta pudesse ser concluída. É com enorme gratidão e estima que expresso aqui o meu apreço por todos os que de alguma forma estiveram envolvidos neste projeto.

Primeiramente agradeço a DEUS, por me dar força e principalmente sabedoria para tomar as decisões certas nos momentos mais difíceis.

A toda a minha família por torcer e sempre acreditar em mim. Em especial minha Mãe Maria Madalena, pelo esforço e exemplo de vida. Aos meus irmãos Marcone, Eduardo e Ana Cláudia pelos conselhos, incentivos e principalmente por me escutarem nos momentos difíceis.

A Titia Pat por ter me dado a oportunidade de dar inicio a meu curso de mestrado, pelo carinho e apoio.

A meu grande amor Evely, os mais profundos agradecimentos por suas lições de esperança; sempre repetindo palavras essenciais - como amor, crença, compreensão, alegria, otimismo - infundiram-me a confiança necessária para realizar os meus sonhos. A agradeço também pelo ombro amigo, pelo amor, carinho e principalmente por saber compreender os momentos importantes que estive ausente.

A meu orientador Luiz Satoru, o meu muito obrigado pela confiança, paciência e por compartilhar seus conhecimentos.

A meu co-orientador Carlos Alberto Martinhon, o muitíssimo obrigado pelas ótimas idéias, pelas revisões de texto, paciência, oportunidade e orientação.

A todos os professores do Instituto de Computação (IC) que participaram de minha formação.

Aos colegas de mestrado, pelos "infinitos"*helps*, pela ótima convivência e por tornarem minha estadia em Niterói muito mais agradável.

A minha família *"Mouseville"*, em especial meus grandes amigos Eduardo, Jonivan, Rone, Áthila e Sidney pelo companheirismo e ótima convivência. Cada um de vocês teve sua parcela de responsabilidade nesta conquista.

Sem dúvida agradecer a todos que me ajudaram nesta longa e árdua caminhada não é uma tarefa fácil. Sei que há tantas outras pessoas que de uma forma ou de outra me ajudaram e muito, mas infelizmente citar a todos é quase que impossível. Então a todos vocês o meu muitíssimo obrigado.

Resumo

Este trabalho aborda uma extensão do *Problema do Caixeiro Viajante - PCV* denominado *Problema do Caixeiro Viajante Branco e Preto-PCVBP*.

O $PCVBP$ é definido sobre um grafo G (orientado ou não), de maneira que o conjunto de vértices associado está particionado em vértices brancos e pretos. O objetivo é encontrar um caminho hamiltoniano de custo mínimo sujeito a restrições adicionais de *Cardinalidade* e *Comprimento*. Estas restrições estão relacionadas respectivamente, ao número de vértices brancos (cardinalidade) e a distância total percorrida (comprimento) entre dois vértices pretos consecutivos em uma solução viável.

As principais aplicações deste problema estão nas áreas de telecomunicação e escalonamento de operações aéreas que necessitam de manutenção de conexões.

Neste trabalho introduzimos formulações matemáticas para o $PCVBP$ assimétrico e simétrico, propomos novas heurísticas de construção e busca local que são utilizadas junto as metaheurísticas $GRASP$, VNS e VND. Resultados computacionais comprovam a viabilidade dos métodos propostos.

Palavras Chaves: $GRASP$, Heurísticas, Metaheurísticas, Problema do Caixeiro Viajante, Regras de Redução, VND, VNS.

Abstract

This work describes a generalization of the *Traveling Salesman Problem - TSP* called *Black and White Traveling Salesman Problem - BWTSP*.

The $BWTSP$ is defined in a graph G (oriented or not), in a such way that the associated vertex set is partitioned into black and white vertices. The objective is to find a shortest hamiltonian tour subject to *Cardinality* and *Length* constraints. These constraints are related to the number of white vertices (cardinality) and the total distance (length) between two consecutive black vertices in a feasible solution.

The main applications of this problem are found in the telecomunication area and the scheduling of airline operations that incorporate maintenance conections.

In this work we introduce mathematical formulations for asymmetrical and symmetrical $BWTSP$, we propose news construction and local search algorithms that are used in the metaheuristics $GRASP$, VNS and VND. Computational results show the viability of the proposed methods.

Key Words: $GRASP$, Heuristics, Metaheuristics, The Traveling Salesman Problem, Reduction Rules, VND, VNS.

Glossário

PCV	:	Problema do Caixeiro Viajante;
PCVBP	:	Problema do Caixeiro Viajante Branco e Preto;
GRASP	:	*Greedy Randomized Adaptative Search Procedure*;
VNS	:	*Variable Neighborhood Search*;
VND	:	*Variable Neighborhood Descent*;
2-OPT	:	Dois Optimal;
GENI	:	*Generalized Insertion Procedure*;
US	:	*Unstringing and Stringing*;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este trabalho tem como objetivo obter soluções aproximadas para uma variante do *Problema do Caixeiro Viajante - PCV* conhecido na literatura como *Problema do Caixeiro Viajante Branco e Preto - PCVBP* [1]. *Bourgeois et. al*[1] em 2001, introduziram o $PCVBP$ e propuseram três heurísticas distintas de construção ($C1$,$C2$ e $C3$) baseadas na heurística $GENIUS$ [2], além de uma heurística de viabilização (F) e uma busca local.

O $PCVBP$ é definido em um grafo G (orientado ou não), onde o conjunto de vértices é particionado em conjuntos disjuntos formados por vértices brancos e pretos. O objetivo é encontrar um caminho hamiltoniano de menor distância sujeito às restrições adicionais de *Cardinalidade* e *Comprimento*. Estas restrições estão relacionadas respectivamente, ao maior número de vértices brancos (representado por Q) e a maior distância percorrida (representada por L) entre dois vértices pretos consecutivos em uma solução viável.

O $PCVBP$ é NP-Dificil já que, para $Q = L = \infty$ ele se reduz ao *Problema do Caixeiro Viajante* reconhecidamente NP-Difícil [3]. Isto significa que uma solução exata obtida em tempo polinomial só será possível se $P = NP$.

Problemas desta natureza são comumente abordados através de *heurísticas*. Basicamente, nas técnicas heurísticas procuramos por boas soluções (próximas da ótima) a um custo computacional razoável sem, no entanto, poder garantir a otimalidade, ao contrário do que ocorre com os métodos exatos. A grande desvantagem das heurísticas tradicionais de construção reside na dificuldade de fugir de ótimos locais, ainda distantes de um ótimo global. Isto deu origem à outra metodologia chamada de *metaheurística*, que possui ferramentas que possibilitam sair destes ótimos locais, permitindo a busca em regiões mais promissoras ainda não pesquisadas. O grande desafio é produzir, em tempo computacional aceitável, soluções que sejam tão próximas quanto possível de uma solução ótima.

Dentre os procedimentos enquadrados como metaheurísticas que surgiram ao longo das últimas décadas, destacam-se: Algoritmos Genéticos [4], *Simulated Annealing* [5], Busca Tabu [6], *Greedy Randomized Adaptative Search Procedure* ($GRASP$) [7], Colônia de Formigas [8], *Variable Neighborhood Search* (VNS) [9], entre outros.

De nosso conhecimento, só existem propostas de três heurísticas convencionais de construção, uma busca local (2-Optimal adaptada ao problema) e uma heurística de viabilização para o $PCVBP$ [1]. Desta forma, propomos o uso das metaheurísticas $GRASP$, VNS e VND com o intuito de obter limites superiores de melhor qualidade para este problema.

O restante deste trabalho está organizado da seguinte forma: no Capítulo 2 é apresentado uma definição formal para o $PCVBP$ e os principais resultados da literatura, no Capítulo 3 é descrito as metaheurísticas $GRASP$, VNS e VND. As formulações matemáticas para o $PCVBP$ (assimétrico e simétrico) e as heurísticas propostas neste trabalho são descritas no Capítulo 4. No Capítulo 5 são apresentados alguns detalhes de implementação. No Capítulo 6 são ilustrados os testes computacionais e as análises dos resultados obtidos. Finalmente apresentamos as conclusões e propostas para trabalhos futuros no Capítulo 7.

Capítulo 2

O Problema do Caixeiro Viajante Branco e Preto

O Problema do Caixeiro Viajante Branco e Preto - PCVBP foi abordado em 2001 por *Bourgeois et. al* [1] e pode ser definido por um grafo direcionado $G = (V, A)$ (não direcionado $G = (V, E)$) onde $V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ representa um conjunto de vértices, $|V| = n + 1$ e $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ um conjunto de arcos de G ($E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$ um conjunto de arestas de G) e cada arco (aresta) está associado a um custo $d_{ij} \geq 0$. O conjunto de vértices é particionado nos subconjuntos disjuntos B e P respectivamente, onde B é composto por vértices brancos, P por vértices pretos onde $|P| \geq 2$. O $PCVBP$ consiste em determinar um circuito (ciclo) hamiltoniano de custo mínimo em G obedecendo às restrições do PCV e sujeito às seguintes restrições adicionais:

- *Cardinalidade*: O número de vértices brancos entre dois vértices pretos consecutivos em uma solução viável não pode ser superior a um número inteiro positivo Q (*Figura 2.1*).
- *Comprimento*: a distância entre dois vértices pretos consecutivos em uma solução viável não deve exceder um número positivo L (*Figura 2.1*).

Como já dito anteriormente o $PCVBP$ é um problema NP- Difícil. Além disso, mesmo garantir que o conjunto de soluções viáveis é não vazio, pode ser uma tarefa complicada, entretanto, caso uma das condições abaixo seja satisfeita, garantimos diretamente que o $PCVBP$ não possui solução viável.

- $\boldsymbol{C_1}$**:** $|B| > Q.|P|$: Se o número de vértices brancos for maior que o número de

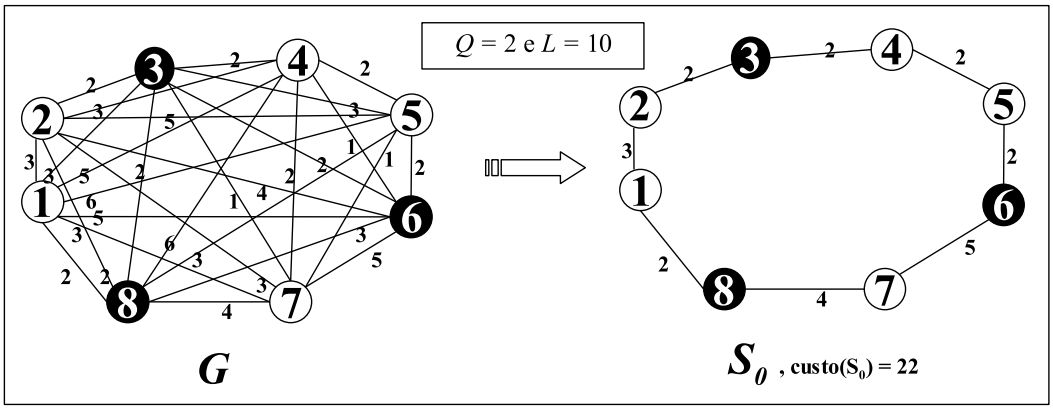


Figura 2.1: Exemplo de uma solução que satisfaz as restrições de *Cardinalidade* e *Comprimento*.

vértices pretos multiplicado por Q, isto significa que a restrição de cardinalidade não pode ser satisfeita.

- **C_2:** $d_{ij_1(i)} + d_{ij_2(i)} > L$, para todo $v_i \in B$ e onde $v_{j_1(i)}$ e $v_{j_2(i)}$ são os dois vértices pretos mais próximos de v_i. Isto significa que não existe um caminho que passe por v_i que seja capaz de satisfazer a restrição de comprimento L.

Note que a condição de inviabilidade C_2 só será aplicada se a desigualdade triangular for satisfeita.

2.1 Heurísticas da Literatura

A seguir serão apresentadas as heurísticas existentes na literatura para o $PCVBP$. Em [1], os autores propõem cinco heurísticas sendo que, as três primeiras são heurísticas de construção e visam determinar uma solução inicial viável ou mesmo inviável para o $PCVBP$. A quarta é uma heurística de viabilização. A quinta e última, busca melhorar as soluções viáveis geradas pelas heurísticas anteriores através de uma exploração de vizinhança.

2.1.1 Heurísticas de Construção

As heurísticas de construção propostas em [1] são baseadas essencialmente na heurística $GENIUS$ [2]. O $GENIUS$ foi proposto por *Gendreau et. al* [2] em 1991 para o PCV e é composto por duas fases. A primeira chamada de $GENI$(*Generalized Insertion Procedure*)

constitui a fase de construção da solução, a segunda chamada de US(*Unstringing and Stringing*) é a fase de refinamento da solução obtida na primeira fase.

Inicialmente a fase $GENI$ considera uma rota parcial do PCV composta por três vértices quaisquer e a cada iteração um novo vértice v é inserido.

O principal aspecto do procedimento $GENI$, que o diferencia dos demais algoritmos de inserção do PCV, é que a inserção do vértice v não é feita necessariamente entre dois vértices consecutivos da rota parcial, embora estes sejam considerados. Um vértice v é inserido levando em consideração sua *vizinhança*, ou seja, $\forall v_i \in V$, define-se uma vizinhança $N_{p(v_i)}$ como sendo os p vértices, pertencentes à rota parcial, mais próximos de v_i, onde p é um parâmetro de entrada. Se menos que p vértices constituem a rota corrente, então todos eles são vizinhos de v_i.

Considere para uma dada orientação da rota os vértices v_{i+1} e v_{i-1} representando respectivamente, o predecessor e o sucessor de v_i.

A inserção de um vértice v entre os vértices v_i e v_j pode ser efetuada de duas maneiras:

Inserção do Tipo I - Define-se para uma determinada orientação da rota: $v_i \in N_{p(v)}$, $v_j \in N_{p(v)}$ e $v_i \neq v_j$; $v_k \in N_{p(v_{i+1})}$, $v_k \neq v_i \neq v_j$ e v_k pertence ao caminho de v_j para v_i. A inserção do vértice v na a rota implica na remoção das arestas (v_i, v_{i+1}), (v_j, v_{j+1}), (v_k, v_{k+1}) e inclusão das arestas (v_i, v), (v, v_j), (v_{i+1}, v_k) e (v_{j+1}, v_{k+1}). Veja *Figura 2.2.*

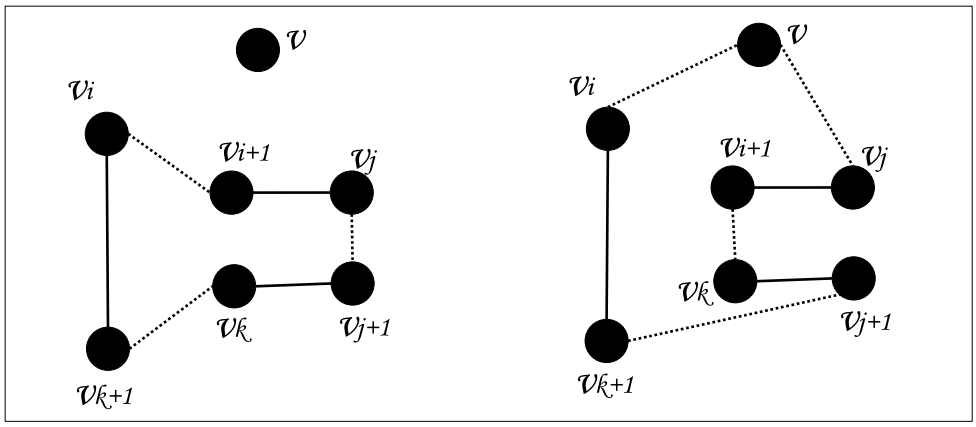


Figura 2.2: Insersão Tipo I (Sentido Horário)

Inserção do Tipo II - Define-se para uma determinada orientação da rota: $v_i \in N_{p(v)}$; $v_j \in N_{p(v)}$ e $v_j \neq v_i$; $v_k \in N_{p(v_{i+1})}$, $v_k \neq v_j, v_k \neq v_{j+1}$ e v_k pertence ao caminho de v_j para v_i; $v_l \in N_{p(v_{j+1})}, v_l \neq v_i, v_l \neq v_{i+1}$ e v_l pertence ao caminho de v_i para v_j. A inserção do vértice v na a rota implica na remoção das arestas (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}), (v_{k-1}, v_k) e inclusão das arestas (v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}) e (v_{i+1}, v_k).

Veja *Figura 2.3.*

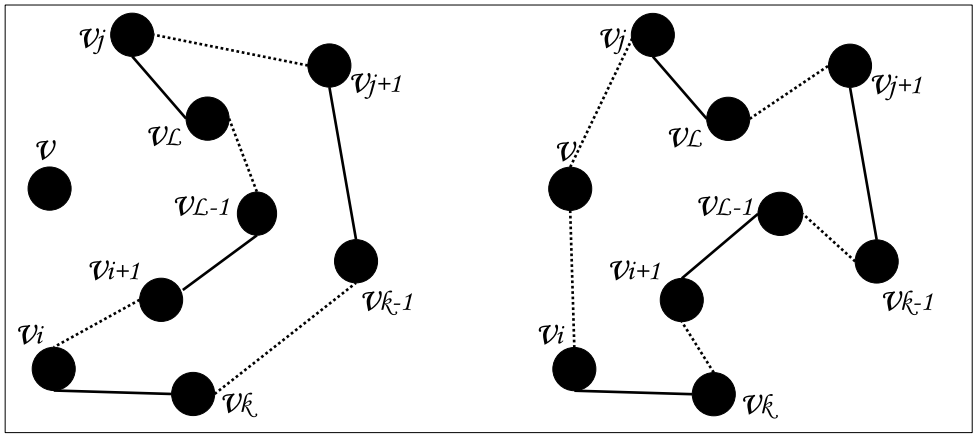


Figura 2.3: Inserção Tipo II (Sentido Horário)

O procedimento $GENI$ considera as duas possíveis orientações na rota para cada inserção, sendo assim, o número de possíveis escolhas para v_i, v_j, v_k, v_l é da ordem de n^4. Para limitar o número de possibilidades, escolhe-se um valor p pequeno reduzindo para $O(p^4)$ o total de combinações dos quatro vértices. O procedimento $GENI$ é descrito pelo *Algoritmo 1.*

Algoritmo 1 GENI

[Passo 1]: Criar uma rota parcial contendo três vértices escolhidos arbitrariamente. Inicializar os p vizinhos para todos os vértices.
[Passo 2]: Escolher arbitrariamente um vértice v que não faça parte da rota. Implementar o procedimento de inserção que produz o menor incremento no custo da solução, levando em consideração os dois tipos de inserção e as duas possíveis orientações da rota. Atualizar os p vizinhos para todos os vértices;
[Passo 3]: Se todos os vértices fazem parte da rota, então pare, senão volte ao Passo 2.

A fase posterior ao $GENI$ é a fase US, cujo objetivo é refinar a solução construída pelo procedimento $GENI$. O procedimento US consiste em percorrer uma solução viável removendo cada vértice v_i e o reinserindo usando o procedimento $GENI$. A remoção de v_i é efetuada de duas maneiras e considerando as duas orientações da rota.

Remoção do Tipo I - Define-se para uma determinada orientação da rota: $v_j \in N_{p(v_{i+1})}$; $v_k \in N_{p(v_{i-1})}$ e v_k pertence ao caminho de v_{i+1} a v_{j-1}. A remoção do vértice v_i da rota implica na remoção das arestas (v_{i-1}, v_i), (v_i, v_{i+1}), (v_k, v_{k+1}), (v_j, v_{j+1}) e a inserção das arestas (v_{i-1}, v_k), (v_{i+1}, v_j) e (v_{k+1}, v_{j+1}). *Figura 2.4.*

Remoção do Tipo II - Define-se para uma determinada orientação da rota: $v_j \in N_{p(v_{i+1})}$; $v_k \in N_{p(v_{i-1})}$ e v_k pertence ao caminho de v_{j+1} a v_{i-2}; $v_l \in N_{p(v_{k+1})}$ no caminho

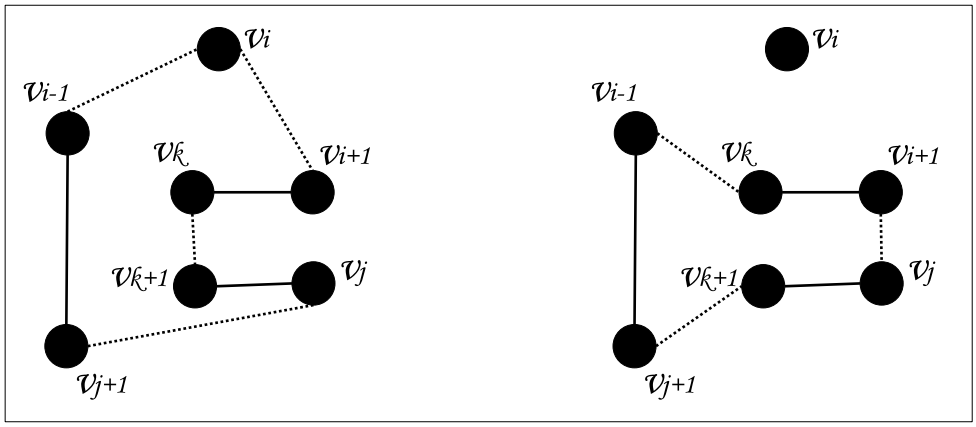


Figura 2.4: Remoção Tipo I (Sentido Horário)

de v_j a v_{k-1}. A remoção do vértice v_i da rota implica na remoção das arestas (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_l, v_{l+1}), (v_k, v_{k+1}) e na inserção das arestas (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j) e (v_l, v_{k+1}). *Figura 2.5.*

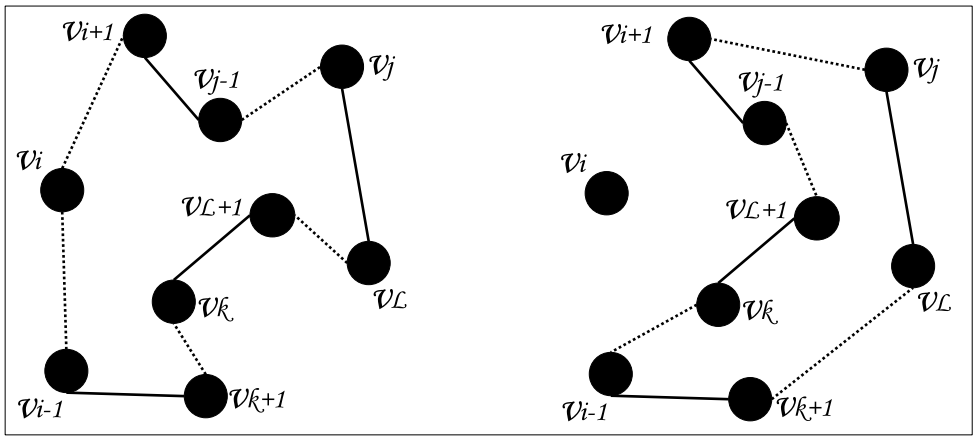


Figura 2.5: Remoção Tipo II (Sentido Horário)

O procedimento US é descrito pelo *Algoritmo 2.*

Considere para o *Algoritmo 2* as soluções s, s' e s^* como sendo respectivamente, solução inicial, solução temporária e melhor solução encontrada. Considere ainda $custo(s)$ uma função que retorna o custo de uma solução s e n o número total de vértices.

A seguir serão apresentadas as heurísticas de construção $C1$, $C2$ e $C3$ propostas em [1].

2.1.1.1 Heurística C1

A heurística $C1$ despreza as restrições de comprimento e cardinalidade e trata o $PCVBP$ como se fosse o PCV. Para construir uma boa solução para o PCV é utilizada a heurística

Algoritmo 2 US(s)

1: $s^* \leftarrow s$;
2: $s' \leftarrow s$;
3: $i \leftarrow 1$;
4: **enquanto** (i $\leq$ n) **faça**
5: Aplique o procedimento de remoção e inserção considerando os dois possíveis tipos e as duas possíveis orientações para o i-ésimo vértice v_i pertencente à solução s. $s' \leftarrow$ resultado da remoção e reinserção.
6: **se** ($custo(s') < custo(s^*)$) **então**
7: $s^* \leftarrow s'$;
8: $s \leftarrow s^*$;
9: $i \leftarrow 1$;
10: **senão**
11: $i \leftarrow i + 1$;
12: **fim se**
13: **fim enquanto**
14: **retorne** s^*;

$GENIUS$. Ao final da heurística $C1$ é produzida uma solução que pode ser ou não viável ao $PCVBP$, já que pode violar tanto a restrição de comprimento quanto a restrição de cardinalidade.

2.1.1.2 Heurística C2

A heurística $C2$ leva em consideração tanto a restrição de cardinalidade quanto a restrição de comprimento. Para gerar uma solução viável para o $PCVPB$, $C2$ inicia aplicando sobre os vértices brancos o $GENIUS$, obtendo desta forma um ciclo somente com os vértices brancos.

Em seguida é aplicado um *procedimento de corte* onde um vértice branco v_r do ciclo parcial corrente é escolhido aleatoriamente, v_r é inserido em um conjunto R (conjunto dos vértices que iniciam a formação de cadeias brancas), e seguindo tanto o sentindo horário quanto anti-horário, cadeias de vértices brancos são construídas a partir do ciclo de vértices brancos obedecendo-se às seguintes regras:

1. O número de vértices brancos pertences a uma cadeia não deve exceder Q.
2. O comprimento da cadeia não deve exceder $L - \ell$. Onde ℓ é duas vezes a distância média de um vértice branco a um vértice preto.

Se o número de cadeias produzidas em cada direção (sentido horário e anti-horário) for superior ao número de vértices pretos $|P|$, a heurística é abortada (a heurística neste

caso não consegue construir uma solução inicial), caso contrário, uma nova matriz D' com distâncias d'_{ij} é construída levando-se em consideração todos os vértices e as seguintes regras:

$$d'_{ij} = \begin{cases} d_{ij} & \text{se } v_i \text{ ou } v_j \in P \\ \infty & \text{se } v_i, v_j \in B \text{ e pertencerem a cadeias diferentes} \\ -\infty & \text{se } v_i, v_j \in B \text{ e pertencerem a mesma cadeia.} \end{cases}$$

Sobre a matriz transformada D' a heurística $GENIUS$ então é aplicada levando em consideração todos os vértices (brancos e pretos) para a construção de uma solução para o PCV. Enquanto as soluções produzidas pela heurística $GENIUS$ usando a matriz D' são sempre viáveis quanto à restrição de cardinalidade, a restrição de comprimento pode ser desrespeitada. Isto acontece quando uma cadeia (trecho da solução iniciada por um vértice preto e terminada com um vértice preto) excede L. Se a cadeia inviável contém no mínimo dois vértices brancos, algumas medidas podem ser tomadas na tentativa de torná-la viável, pois se escolhermos um novo v_r (vértices branco) pertencente a uma cadeia inviável como ponto partida para que a heurística $C2$ seja reinicializada, soluções viáveis poderão ser construídas. A escolha do novo v_r é efetuada da seguinte maneira: dada uma aresta branca $(v_s, v_t) \in A$ (onde $v_s, v_t \in B$) e pertencente à cadeia inviável, fazemos $v_r \longleftarrow v_s$ se $v_s \notin R$, fazemos $v_r \longleftarrow v_t$ se $v_s \in R$ e $v_t \notin R$. Se $v_s, v_t \in R$ a heurística é abortada, caso contrário $R \longleftarrow R \cup \{v_r\}$ e a heurística $C2$ é reinicializada a partir do procedimento de corte.

2.1.1.3 Heurística C3

A heurística $C3$ também leva em consideração as restrições de cardinalidade e comprimento. Assim como as outras heurísticas de construção, $C3$ também utiliza a heurística $GENIUS$.

Inicialmente a heurística $C3$ é aplicada sobre os vértices brancos através do procedimento $GENIUS$. Sobre o ciclo obtido, um vértice $v_s \in B$ é escolhido aleatoriamente. O próximo passo é associar a v_s o vértice preto mais próximo (v_r), adicionando-o onde cause o menor incremento do custo da solução (à direita ou à esquerda de v_s). A partir de v_r o ciclo é percorrido (tanto no sentido horário quanto no sentido anti-horário) enquanto as restrições de cardinalidade e comprimento possam ser satisfeitas. Quando Q e L não poderem ser mais satisfeitos, o vértice preto mais próximo do último vértice branco percorrido deve ser inserido de forma que a cadeia (iniciada em um vértice preto e terminada

em um vértice preto) formada respeite as restrições de cardinalidade e comprimento. Esta heurística se diferencia de $C2$; entre outras coisas por não trabalhar com a matriz D' e não garantir que as restrições de comprimento e/ou cardinalidade sejam satisfeitas.

A heurística $C3$ aborta quando é impossível iniciar uma nova cadeia. Se o último vértice branco é atingido e vértices pretos não foram inseridos à solução, estes são inseridos ao final da cadeia de forma a causar o menor incremento possível à solução.

2.1.2 Heurística de Viabilização F

De acordo com as heurísticas de construção vistas anteriormente pode-se ter ao final da execução, uma solução inviável produzida por $C1$ ou duas soluções inviáveis produzidas por $C2$ e $C3$. Note que $C2$ e $C3$ geram uma solução associada a cada orientação.

A heurística F tem como objetivo eliminar a inviabilidade gerada determinando uma solução viável caso seja possível. Se a restrição de comprimento é violada isto implica que alguma(s) cadeia(s) (iniciada e terminada em um vértice preto) possui(em) comprimento maior do que L. Esta inviabilidade é chamada de *inviabilidade preta*. A inviabilidade preta pode ser eliminada pela inserção de um vértice preto entre dois pontos da cadeia que viola L (*Figura 2.6*). Quando a restrição de cardinalidade é violada denotamos esta situação de *inviabilidade branca*. Esta pode ser resolvida pela remoção de vértices brancos do interior da cadeia e reinserindo-os em cadeias que possuem no máximo $Q - 1$ vértices brancos (*Figura 2.7*).

A heurística F sempre tenta eliminar inicialmente a inviabilidade preta e posteriormente a inviabilidade branca. Desta forma, é importante ressaltar que na tentativa de se eliminar a inviabilidade preta, esta pode causar inviabilidade branca, mas a recíproca não deve ocorrer. Outro aspecto importante na eliminação da inviabilidade preta é que a inserção de um vértice preto em uma cadeia que viole L implica na remoção de um vértice preto pertencente à outra cadeia. Esta remoção não deve causar uma nova inviabilidade preta e para evitar a formação de ciclos um vértice só pode ser removido uma única vez.

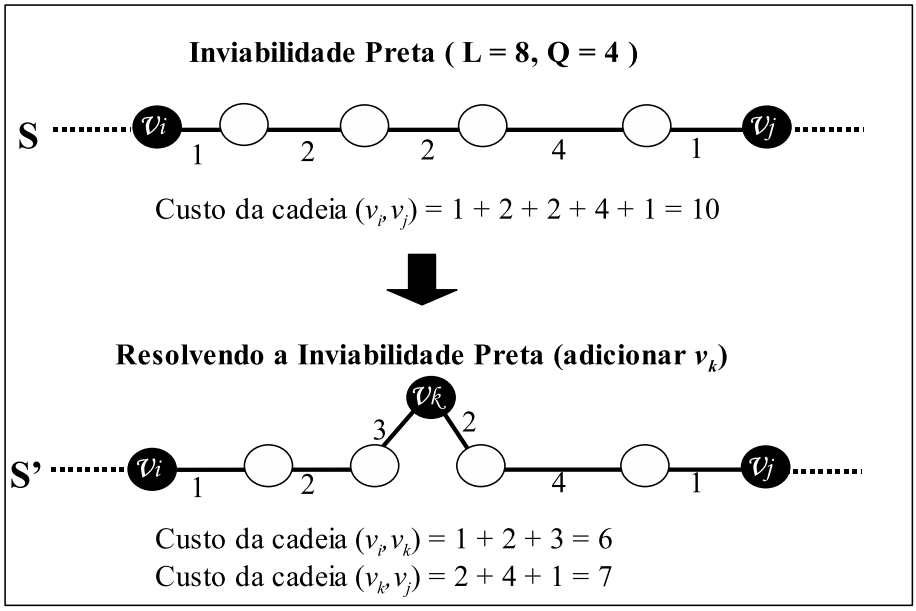


Figura 2.6: Inviabilidade Preta

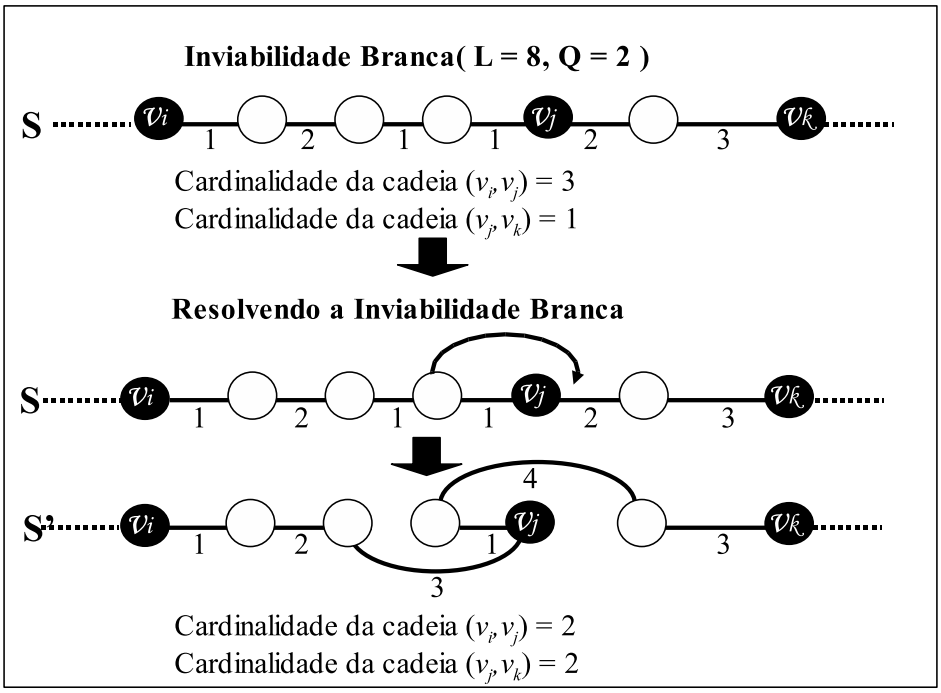


Figura 2.7: Inviabilidade Branca

2.1.3 Heurística de Refinamento

A heurística de refinamento explorada em [1] é a Heurística $2 - Optimal$ $(2 - OPT)$, proposta inicialmente em [10]. O $2 - OPT$ proposto é aplicado somente sobre as soluções viáveis e busca melhorar uma solução preservando sua viabilidade através da exploração

de sua vizinhança. O $2-OPT$ explora a vizinhança da solução corrente efetuando todas as possíveis trocas de arestas entre vértices não adjacentes. O *Algoritmo 3* descreve a heurística de refinamento proposta em [1].

Algoritmo 3 2-OPT

1: Seja s_0 uma solução inicial;
2: $s^* \leftarrow s_0$;
3: $C \leftarrow$ lista de todos os subconjuntos de 2 arestas não adjacentes de s_0;
4: **enquanto** $|C| > 0$ **faça**
5: $Z \leftarrow$ subconjunto formado por um elemento de C;
6: Remover as arestas $\in Z$ de s_0 gerando s';
7: Reconstruir s';
8: **se** $(custo(s') < custo(s^*))e(solucao\_viavel(s'))$ **então**
9: $s^* \leftarrow s'$;
10: **fim se**
11: $C \leftarrow C - Z$;
12: **fim enquanto**
13: **retorne** s^*;

Para o *Algoritmo 3* considere s_0 uma solução inicial viável, s^* a melhor solução encontrada, Z um sub-conjunto de C formado por um par de arestas não adjacentes. Considere ainda $custo(s)$ uma função que retorna o custo da solução s e $solucao\_viavel(s)$ uma função que retorna verdadeira caso s respeite as restrições de cardinalidade e comprimento, caso contrário falso.

As *Figuras 2.8 a 2.13* ilustram o *Algoritmo 3*.

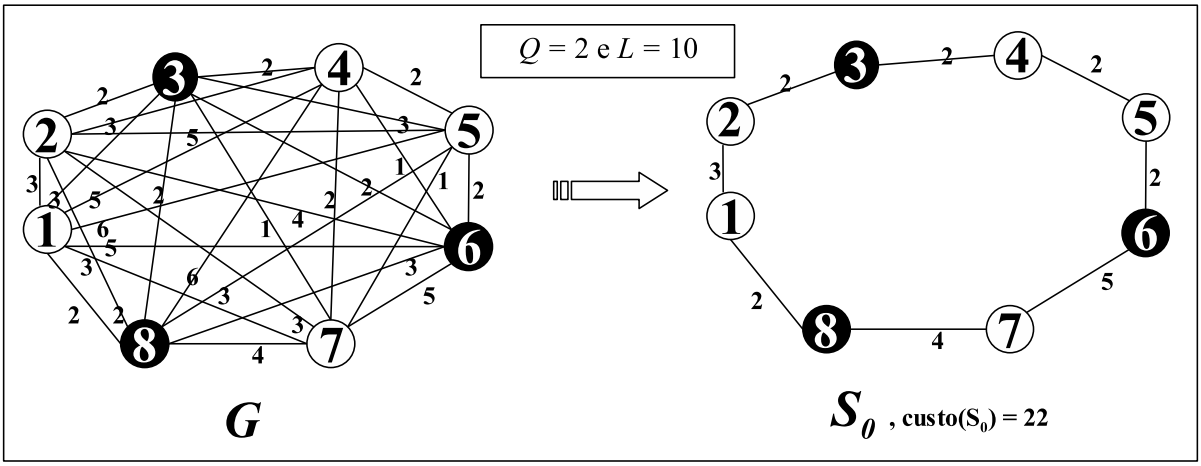
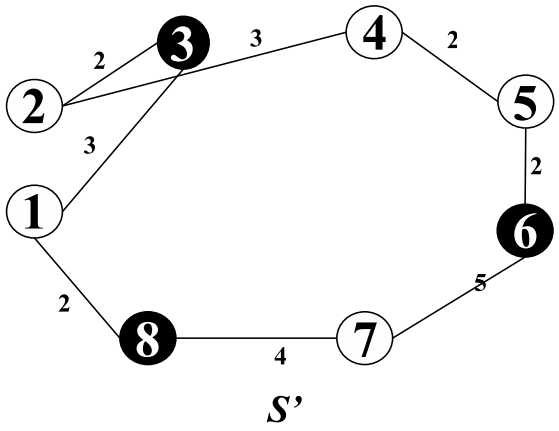


Figura 2.8: Execução 2-OPT, Solução inicial S_0.

C = { (1,2) (3,4); (1,2) (4,5); (1,2) (5,6); (1,2) (6,7); (1,2) (7,8); (2,3) (4,5); (2,3) (5,6); (2,3) (7,8); (2,3) (8,1); (3,4) (5,6); (3,4) (6,7); (3,4) (7,8); (4,5) (6,7); (4,5) (7,8); (4,5) (8,1); (5,6) (7,8); (5,6) (8,1); (6,7) (8,1) }

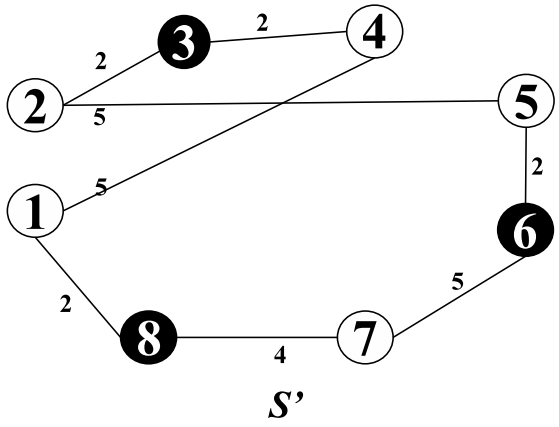


Movimento descartado (**Z**={(1,2) (3,4)}), desrespeita a restrição de cardinalidade.

C = C – Z

Figura 2.9: Execução 2-OPT, primeira iteração

C = { (1,2) (4,5); (1,2) (5,6); (1,2) (6,7); (1,2) (7,8); (2,3) (4,5); (2,3) (5,6); (2,3) (7,8); (2,3) (8,1); (3,4) (5,6); (3,4) (6,7); (3,4) (7,8); (4,5) (6,7); (4,5) (7,8); (4,5) (8,1); (5,6) (7,8); (5,6) (8,1); (6,7) (8,1) }

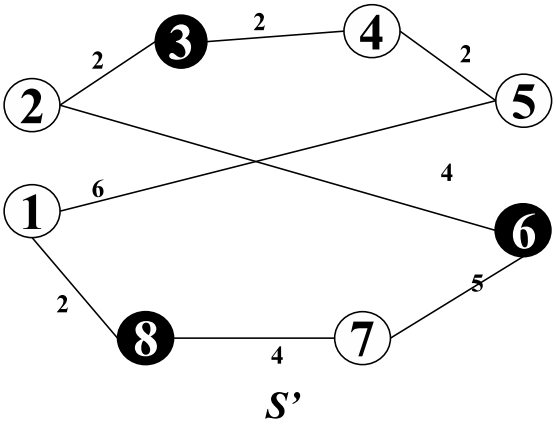


Movimento descartado (**Z**={(1,2) (4,5)}), custo(S') = 27.

C = C – Z

Figura 2.10: Execução 2-OPT, segunda iteração

C = { (1,2) (5,6); (1,2) (6,7); (1,2) (7,8); (2,3) (4,5); (2,3) (5,6); (2,3) (7,8); (2,3) (8,1); (3,4) (5,6); (3,4) (6,7); (3,4) (7,8); (4,5) (6,7); (4,5) (7,8); (4,5) (8,1); (5,6) (7,8); (5,6) (8,1); (6,7) (8,1) }

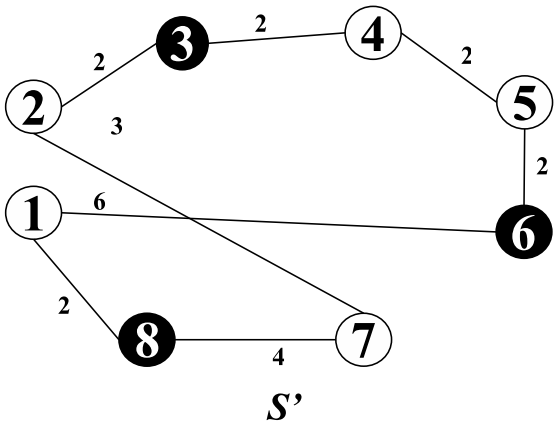


Movimento descartado (**Z**={(1,2) (5,6)}), desrespeita a restrição de cardinalidade e comprimento.

C = C – Z

Figura 2.11: Execução 2-OPT, terceiera iteração

C = { (1,2) (6,7); (1,2) (7,8); (2,3) (4,5); (2,3) (5,6); (2,3) (7,8); (2,3) (8,1); (3,4) (5,6); (3,4) (6,7); (3,4) (7,8); (4,5) (6,7); (4,5) (7,8); (4,5) (8,1); (5,6) (7,8); (5,6) (8,1); (6,7) (8,1) }

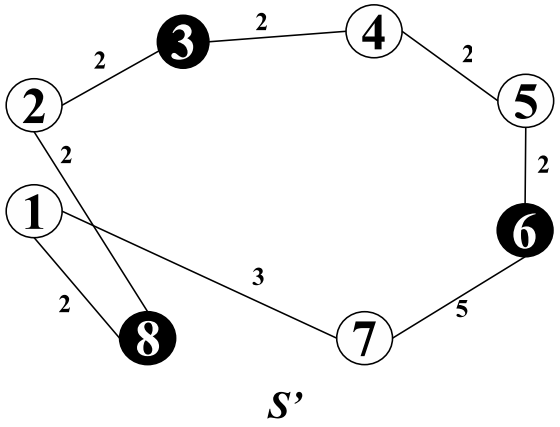


Movimento descartado (**Z**={(1,2) (5,6)}), custo(S') = 22.

C = C – Z

Figura 2.12: Execução 2-OPT, quarta iteração

C = { (1,2) (7,8); (2,3) (4,5); (2,3) (5,6); (2,3) (7,8); (2,3) (8,1); (3,4) (5,6); (3,4) (6,7); (3,4) (7,8); (4,5) (6,7); (4,5) (7,8); (4,5) (8,1); (5,6) (7,8); (5,6) (8,1); (6,7) (8,1) }



Movimento de melhora
S' viável (Z={(1,2) (7,8)}),
custo(S') = 20.

S* ← S'.

C = C – Z

Enquanto C ≠ ∅ , continue .....

Figura 2.13: Execução 2-OPT, quinta iteração

Capítulo 3

Metaheurísticas

Há vários fatores que influenciam a escolha de uma técnica para resolução de um determinado problema, como a natureza dos dados, o tamanho do conjunto de entrada, etc. Muitas vezes opta-se pelo uso de heurísticas ou metaheurísticas para auxiliar na obtenção de uma solução de boa qualidade, abrindo-se mão da garantia de se obter uma solução ótima pela diminuição do tempo de processamento.

O uso de heurísticas de construção e buscas locais têm sido freqüentes nas últimas décadas. Um dos problemas que mais recebeu contribuições desta natureza foi o *Problema do Caixeiro Viajante - PCV* [11].

Métodos heurísticos além de permitir a solução de problemas de elevada complexidade em um tempo computacional aceitável, possuem a vantagem de serem flexíveis, adaptando-se às possíveis mudanças nos dados de entrada do problema. Contudo, as heurísticas tradicionais de construção e busca local normalmente não conseguem escapar de ótimos locais muitas vezes ainda distantes de um ótimo global.

A partir dos anos 80, tem-se encontrado na literatura diferentes heurísticas genéricas também conhecidas como metaheurísticas; métodos estes, que incorporam procedimentos que permitem escapar de ótimos locais e assim procurar ao seu final obter limites de melhor qualidade para o valor ótimo em um problema de otimização.

Dentre as metaheurísticas mais competitivas para o PCV, problema de roteamento e *scheduling*, podemos incluir os Algoritmos Genéticos [4], Busca Tabu [6], $GRASP$ [7], VNS [9]. Todos estes métodos já são bastante usados em problemas de gerar rotas otimizadas mas, os mais explorados são a Busca Tabu [12, 13] e os Algoritmos Genéticos. O $GRASP$ e VNS embora mais recentes, estão entre as heurísticas mais eficientes em diferentes aplicações de otimização [14, 15].

Por estes motivos, neste trabalho optamos pelas metaheurísticas $GRASP$, VND e VNS para a solução aproximada do $PCVBP$.

3.1 GRASP - *Greedy Randomized Adaptative Search Procedure*

A metaheurística $GRASP$ (*Greedy Randomized Adaptative Search Procedure*) é um procedimento multipartida [7, 16, 17], no qual cada iteração consiste em duas fases: construção e busca local. A fase de construção é responsável por gerar uma solução inicial viável, na etapa de busca local, sua vizinhança é investigada até que um ótimo local seja encontrado. Este procedimento é repetido por várias iterações e a melhor solução obtida ao longo de todas as iterações é fornecida como resultado. O *Algoritmo 4* ilustra as principais etapas de um procedimento $GRASP$ aplicado a um problema de minimização.

Algoritmo 4 GRASP (MAX_iterações)

```
1: s* ← ∅;
2: custo(s*) ← ∞;
3: k ⟵ 0;
4: enquanto k ≤ MAX_iterações faça
5:    s ← Construção_GRASP;
6:    s ← Busca_Local(s);
7:    se custo(s*) > custo(s) então
8:       s* ← s;
9:    fim se
10:   k ⟵ k + 1;
11: fim enquanto
12: retorne s*;
```

O *Algoritmo 5* ilustra a fase de construção do $GRASP$. A cada iteração desta fase, o conjunto de elementos candidatos é formado por todos os elementos que ainda não foram incorporados à solução parcial em construção. A seleção do próximo elemento a ser incorporado é determinado pela avaliação de todos os elementos candidatos de acordo com uma função gulosa. Esta função de avaliação representa o aumento incremental na função objetivo devido à incorporação deste elemento na solução corrente. A avaliação dos elementos segundo esta função leva a criação de uma lista restrita de candidatos (LCR) formada pelos melhores elementos, isto é, aqueles que cuja incorporação à solução parcial corrente resulta nos menores custos incrementais (aspecto guloso do algoritmo). O elemento a ser incorporado à solução parcial é então selecionado aleatoriamente dentre aqueles da LCR (aspecto probabilístico do algoritmo). Uma vez que o elemento seleci-

onado foi incorporado à solução parcial, a lista de candidatos é atualizada e os custos incrementais são reavaliados (aspecto adaptativo do algoritmo).

Algoritmo 5 Construção_GRASP

1: $s \leftarrow \varnothing$;
2: Inicializar o conjunto de candidatos C;
3: Construir uma lista restrita de Candidatos (LCR);
4: **enquanto** s não for uma solução completa **faça**
5: Selecionar aleatoriamente um elemento $t \in LCR$;
6: $s \leftarrow s \cup t$;
7: Reavaliar os custos incrementais dos elementos da LCR;
8: **fim enquanto**
9: **retorne** s;

As soluções geradas por uma construção gulosa *randomizada* não representam necessariamente um ótimo local ou global. Portanto é fortemente recomendado a utilização de uma técnica de refinamento. A fase de busca local procura melhorar a solução obtida na etapa gulosa. Algoritmos de busca local funcionam de maneira iterativa, substituindo sucessivamente a solução corrente por outra melhor em sua vizinhança. A busca local termina quando nenhuma solução que apresente melhora é encontrada na vizinhança da solução corrente. O algoritmo básico de busca local iniciado a partir de uma solução s construída na primeira fase e usando uma vizinhança $N(.)$ é apresentada pelo *Algoritmo 6*.

Algoritmo 6 Busca Local

1: **enquanto** s não for uma solução localmente ótima **faça**
2: Obter $s' \in N(s)$ tal que $custo(s') < custo(s)$;
3: $s \leftarrow s'$
4: **fim enquanto**
5: **retorne** s;

3.1.1 Construção da Lista Restrita de Candidatos - *LCR*

Segundo [17] existem duas estratégias básicas usadas para construir a LCR. A primeira delas é baseada em *cardinalidade*, onde, para um valor inteiro k pré-estabelecido, coloca-se na LCR os k melhores elementos na lista de candidatos. A outra abordagem é um esquema baseado no *valor* associado a cada elemento candidato. Seja $f(t)$ o custo incremental associado à incorporação do elemento t a solução em construção, C o conjunto dos elementos candidatos a fazerem parte da LCR, $c_{max} = max\{f(t)|t \in C\}$, $c_{min} = min\{f(t)|t \in C\}$ e um parâmetro $\alpha \in [0, 1]$. Em um problema de minimização, uma LCR baseadas em

valores será determinada por $LCR = \{t \in C | f(t) \leq c_{min} + \alpha(c_{max} - c_{min})\}$. O caso $\alpha = 0$ corresponde ao algoritmo guloso puro, enquanto que, para $\alpha = 1$, são construídas soluções aleatórias. Analogamente em um esquema baseado em cardinalidade, o caso $k = 1$ corresponde a um algoritmo puramente guloso. Já para $k = |C|$, são construídas soluções aleatórias. Assim, os parâmetros α e k determinam se a fase construtiva da metaheurística $GRASP$ será semelhante a um algoritmo de construção puramente guloso ou um algoritmo aleatório. O *Algoritmo 7* é um refinamento do *Algoritmo 5*. Ele mostra que o parâmetro α controla a intensidade dos aspectos guloso e *randomizado* do algoritmo.

De acordo com [18] a maioria das implementações $GRASP$ encontradas na literatura usam algum tipo de LCR baseada no valor da função gulosa dos elementos candidatos. As primeiras implementações deste método utilizavam valores fixos para α, que eram determinados através de experimentos computacionais. Para cada problema tratado, ou até mesmo para cada classe de instâncias, um valor distinto de α era preestabelecido. Uma versão adaptativa do parâmetro α foi proposto em Prais e Ribeiro [19] denotado $GRASP$ reativo.

A metaheurística $GRASP$ pode ser vista como uma técnica repetitiva de amostragem no espaço de busca. Cada iteração produz uma solução que é uma amostra de uma distribuição desconhecida. A média e a variância do valor das soluções obtidas ao longo de $MAX\_iteracoes$ iterações dependem da natureza restritiva da LCR. Por exemplo, se a LCR for restrita a um único elemento, então a mesma solução obtida será produzida ao longo de todas as iterações. A variância da distribuição será nula e a média será igual ao valor da solução gulosa. Se a LCR possuir mais de um elemento, então muitas soluções diferentes serão produzidas e a variância será maior. Como o aspecto guloso desempenha um papel importante neste caso, o valor médio das soluções tende neste caso a ser pior. Entretanto, a melhor solução encontrada poderá ser ao final de muitas iterações ser superior à solução obtida pelo algoritmo puramente guloso.

Prais e Ribeiro [19] estudaram o comportamento de um algoritmo $GRASP$ em relação a diversidade de soluções, à qualidade de soluções e ao tempo computacional, em função do parâmetro α, que restringe a LCR. Eles concluíram que, quanto maior a variância dos valores das soluções encontradas durante a fase de construção, maior será a variância das soluções obtidas após a busca local. A probabilidade de um algoritmo $GRASP$ encontrar uma solução ótima é maior quando a variância das soluções construídas é grande. Porém, neste caso, como a diferença entre o valor médio das soluções construídas e o valor da melhor solução é grande, a busca local precisará em média de um maior tempo computa-

cional. Conseqüentemente o tempo do algoritmo será maior quando a variância dos custos produzidas na fase construtiva for alta. Conclui-se, então, que a escolha apropriada do parâmetro α é crítica no compromisso entre o tempo de processamento e a qualidade da solução.

Algoritmo 7 Construção_GRASP

1: $s \leftarrow \varnothing$;
2: Inicializar o comjunto de ementos candidatos C;
3: Avaliar os custos incrementais $f(t)$ para cada elemento $t \in C$;
4: **enquanto** $(C \neq \varnothing)$ **faça**
5: $c_{min} = min\{f(t)|t \in C\}$;
6: $c_{max} = max\{f(t)|t \in C\}$;
7: $LCR = \{t \in C|f(t) \leq c_{min} + \alpha(c_{max} - c_{min})\}$;
8: Selecionar aleatoriamente um elemento $t \in LCR$;
9: $s \leftarrow s \cup t$;
10: Atualizar o conjunto C de elementos candidatos;
11: Reavaliar o custo incremental $f(t)$ de cada elemento $t \in C$;
12: **fim enquanto**
13: **retorne** s

A seguir serão apresentadas duas técnicas alternativas que podem ser incorporadas a metaheurística $GRASP$.

3.1.2 GRASP Reativo

O $GRASP$ Reativo [19] é uma técnica que incorpora mecanismos de aprendizado na sua fase de construção. Neste caso, o valor do parametro α que define o tamanho da LCR não é fixo. Em vez disto, este valor é selecionado a cada iteração dentre os elementos de uma distribuição discreta de valores possíveis. Esta seleção é guiada pelos valores das soluções encontradas ao longo das iterações precedentes. Uma maneira possível de alcançar este efeito consiste em usar a seguinte regra: Seja $\psi = \{\alpha_1, ..., \alpha_m\}$ o conjunto de valores possíveis de α. As probabilidades associadas à escolha de cada valor são inicializadas com $p_i = 1/m, i = 1, ..., m$. Além disto, seja s^* a melhor solução já encontrada e A_i o valor médio de todas as soluções encontradas utilizando-se $\alpha = \alpha_i, i = 1, ..., m$. As probabilidades de seleção são periodicamente reavaliadas como $p_i = q_i/(\sum_{j=1}^{m} q_j)$, onde $q_i = s^*/A_i$ para $i = 1, ..., m$. O valor de q_i será maior para os valores $\alpha = \alpha_i$ associado às melhores soluções. Maiores valores de q_i correspondem a valores mais apropriados para o parâmetro α. As probabilidades associadas aos valores mais apropriados irão aumentar quando forem reavaliadas. O Enfoque reativo pode levar a melhorias sensíveis em relação ao procedimento $GRASP$ básico, em torno da robustez do algoritmo e a qualidade das

soluções, devido à maior diversidade destas últimas e a menor dependência do ajuste de parâmetros.

3.1.3 Reconexão por Caminhos

Uma das maneiras de se melhorar a qualidade das soluções obtidas pelo $GRASP$ básico é o uso de reconexão por caminhos. A técnica de reconexão por caminhos foi proposta, originalmente, por Glover [20] como uma estratégia de intensificação, explorando trajetórias que conectavam soluções elites obtidas pela *Busca Tabu*. Neste contexto, na busca por melhores soluções são gerados e explorados caminhos no espaço de soluções intermediária entre duas soluções de boa qualidade, denominadas de *soluções base* ou *soluções guia*. Isto é alcançado selecionando-se movimentos que introduzem atributos das soluções guia na solução corrente. Esta técnica pode ser vista como uma estratégia que busca incorporar atributos das soluções de boa qualidade, favorecendo a seleção de movimentos que os contenham. Maiores detalhes deste procedimento pode ser visto em [21].

3.2 VNS - *Variable Neighborhood Search*

O método de Pesquisa em Vizinhança Variável (*Variable Neighborhood Search*, VNS) foi proposto Nenad Mladenovic [22]. Este é um método de busca que consiste em explorar o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança.

Ao contrário de outras metaheurísticas baseadas em métodos de busca local, o VNS não segue uma trajetória, mas sim explora vizinhanças gradativamente mais distantes da solução inicial e focaliza a busca em torno de uma nova solução sempre que um movimento de melhora é realizado.

O método se caracteriza essencialmente por um procedimento de busca local aplicado à solução corrente. Esta rotina de busca local também pode usar diferentes estruturas de vizinhança.

Um algoritmo VNS pode ser descrito conforme o pseudocódigo *Algoritmo 8*. O algoritmo parte de uma solução inicial qualquer e a cada iteração seleciona aleatoriamente um vizinho s' dentro da *k-ésima* vizinhança, $N_{(k)}(s)$ sendo s a solução corrente. Esse vizinho é então submetido a um procedimento de busca local. Se a solução ótima local, s'', for melhor que a solução s corrente, a busca continua a partir de s'' recomeçando da primeira estrutura de vizinhança $N_{(1)}(s)$. Caso contrário, continua a busca a partir da próxima

estrutura de vizinhança $N_{(k+1)}(s)$. Este procedimento é encerrado quando uma condição de parada for atingida, como por exemplo o número máximo de iterações.

Algoritmo 8 VNS

```
1: seja s_0 uma solução inicial;
2: seja r o número de estruturas diferentes de vizinhanças;
3: s ← s_0;
4: enquanto critério de parada não satisfeito faça
5:    k ← 1;
6:    enquanto k ≤ r faça
7:       gere um vizinho qualquer s' ∈ N_(k)(s);
8:       s'' ← Busca Local(s');
9:       se custo(s'') < custo(s) então
10:         s ← s'';
11:         k ← 1;
12:      senão
13:         k ← k + 1;
14:      fim se
15:   fim enquanto
16: fim enquanto
17: retorne s;
```

3.3 VND - *Variable Neighborhood Descent*

O VND é uma variante do VNS, ou mais especificamente um caso particular, também proposto por *Hansen e Mladenovic*[9].

Com o decorrer dos anos *Hansen e Mladenovic* publicaram vários trabalhos relacionados ao VNS, VND e suas variações [23, 24], de acordo com estes trabalhos o VND pode ser descrito de duas formas: *Algoritmo 9* [23] e *Algoritmo 10* [24].

Algoritmo 9 VND

```
1: seja s_0 uma solução inicial;
2: seja r o número de estruturas diferentes de vizinhança;
3: s ← s_0;
4: k ← 1;
5: enquanto k ≤ r faça
6:    encontre o melhor vizinho s' ∈ N_(k)(s);
7:    se custo(s') < custo(s) então
8:       s ← s';
9:    fim se
10:   k ← k + 1;
11: fim enquanto
12: retorne s;
```

Algoritmo 10 VND

```
1: seja s_0 uma solução inicial;
2: seja r o número de estruturas diferentes de vizinhança;
3: s ← s_0;
4: k ← 1;
5: enquanto k ≤ r faça
6:    encontre o melhor vizinho s' ∈ N_(k)(s);
7:    se custo(s') < custo(s) então
8:       s ← s';
9:       k ← 1;
10:   senão
11:      k ← k + 1;
12:   fim se
13: fim enquanto
14: retorne s;
```

Note que o *Algoritmo 9* se diferencia do *Algoritmo 10* por não retornar à primeira vizinhança quando um movimento de melhora da solução é alcançado.

Capítulo 4

Propostas

Neste trabalho introduzimos formulações matemáticas para o $PCVBP$ (simétrico e assimétrico), propomos novas heurísticas de construção e busca local que são utilizadas pelas metaheurísticas $GRASP$, VNS e VND na tentativa de encontrar soluções de melhor qualidade para o problema. As formulações matemáticas são apresentadas na *seção 4.1*, as heurísticas de construção, busca local e metaheurísticas são apresentadas respectivamente nas *seções 4.2.1*, *4.2.2* e *4.2.3*.

4.1 Formulações Matemáticas Propostas

Nesta seção é proposta inicialmente uma formulação matemática para o $PCVBP$ assimétrico e posteriormente uma formulação matemática para o $PCVBP$ simétrico descrevendo-os como um Problema de Programação Linear Inteira. Para se ter uma melhor compreensão dos modelos, as seguintes considerações devem ser feitas: dado um grafo G (orientado ou não) com um conjunto de vértices $V = B \cup P$, onde os vértices são indexados de 0 a n e $|V| = n + 1$, B representa o conjunto de vértices brancos e P representa o conjunto de vértices pretos. Representa-se por ϕ um conjunto de *"cores"* associadas aos arcos (às arestas) de cada caminho entre dois vértices pretos. Dessa forma, todos os arcos (todas as arestas) pertencentes a este caminho deverão ser *"coloridos(as)"* com uma mesma cor. Observe que, o número de caminhos iniciados e terminados em vértices pretos será igual a $|\phi|$. Assume-se ainda, nos casos simétrico e assimétrico, que z_{ij} e x_{ij}^k são variáveis binárias. Assim, se $z_{ij} = 1$, o arco (a aresta) (v_i, v_j) pertence a solução viável, e $z_{ij} = 0$ caso contrário. Se $x_{ij}^k = 1$, o arco (a aresta) (v_i, v_j) de cor k pertencerá a solução viável, e $x_{ij}^k = 0$ caso contrário. No caso não orientado assumimos sempre $i < j$. Assim, o objetivo será encontrar o circuito (ciclo) hamiltoniano de comprimento mínimo e que satisfaça as

restrições de cardinalidade e comprimento.

A seguir serão apresentadas as formulações matemáticas para o $PCVBP$ assimétrico e simétrico respectivamente:

4.1.1 PCVBP Assimétrico

Seja $G = (V, A)$, um grafo orientado com pesos $d_{ij} \geq 0$ associados a cada arco. Considere ainda que a origem é indicada pelo vértice v_0 podendo representar um vértice branco ou preto indistintamente. Dado um arco $(v_i, v_j) \in A$ de custo d_{ij}, dizemos que (v_i, v_j) *parte (outcoming edge)* de $v_i \in A$ e *chega (incoming edge)* a $v_j \in A$.

$$min \sum_{(v_i,v_j)\in A} d_{ij} z_{ij} \tag{4.1}$$

sujeito a:

$$\sum_{k\in\phi} \sum_{v_j\in V/\{v_i\}} x_{ij}^k = 1, \forall v_i \in V \tag{4.2}$$

$$\sum_{k\in\phi} \sum_{v_j\in V/\{v_i\}} x_{ji}^k = 1, \forall v_i \in V \tag{4.3}$$

$$\sum_{v_i\in V/\{v_j\}} x_{ij}^k = \sum_{v_l\in V/\{v_j\}} x_{jl}^k, \forall v_j \in B, \forall k \in \phi \tag{4.4}$$

$$\sum_{(v_i,v_j)\in A} d_{ij} x_{ij}^k \leq L, \forall k \in \phi \tag{4.5}$$

$$\sum_{(v_i,v_j)\in A} x_{ij}^k \leq Q + 1, \forall k \in \phi \tag{4.6}$$

$$\sum_{v_i\in Pe(v_i,v_j)\in A} x_{ij}^k = 1, \forall k \in \phi \tag{4.7}$$

$$\sum_{v_l\in Pe(v_j,v_l)\in A} x_{jl}^k = 1, \forall k \in \phi \tag{4.8}$$

$$\sum_{i=0}^{n} y_{ij} - \sum_{i=0}^{n} y_{ji} = 1, j = 1, 2, ..., n \tag{4.9}$$

$$y_{ij} \leq n z_{ij}, \forall (v_i, v_j) \in A \tag{4.10}$$

$$\sum_{k \in \phi} x_{ij}^{k} = z_{ij}, \forall (v_i, v_j) \in A \tag{4.11}$$

$$x_{ij}^{k}, z_{ij} \in \{0, 1\}, y_{ij} \geq 0, \forall (v_i, v_j) \in A, \forall k \in \phi \tag{4.12}$$

Note que $O(n^2)$ restrições foram geradas.

Na função objetivo, representada por (4.1) desejamos minimizar o somatório do custo dos arcos presentes em uma solução viável. O conjunto de restrições (4.2) garante que apenas um único arco de cor k parta do vértice $v_i \in V$. Analogamente, o conjunto de restrições (4.3) garante que apenas um único arco de cor k chegue ao vértice $v_i \in V$. O conjunto de restrições (4.4) garante que para todo vértice branco ($v_j \in B$) a cor do arco que chega em v_j é igual à cor do arco que parte de v_j. Como no modelo proposto, cada cor $k \in \phi$ está associada a uma cadeia iniciada e terminada em um vértice preto, o somatório de todos os arcos de cor k indica o seu comprimento, logo para atender a restrição de comprimento, este somatório deve ser menor ou igual a L (conjunto de restrições (4.5)). Note que uma cadeia de cor k composta por p vértices, terá $p-1$ arcos e $p-2$ vértices brancos associados. Logo, para satisfazer a restrição de cardinalidade, o número de vértices brancos em uma cadeia de cor k deve ser menor ou igual a Q (conjunto de restrições (4.6)). O conjunto de restrições (4.7) garante que uma cadeia de cor k possua origem em um vértice preto. Analogamente, o conjunto de restrições (4.8) garante que uma cadeia de cor k possui um único destino preto. Os conjuntos de restrições (4.9), (4.10) e (4.12) são restrições de fluxo e garantem a formação de um ciclo hamiltoniano contendo a origem. Note que o conjunto de restrições (4.9) garante a não existência de sub-rotas não passando pela origem. Entretanto, podemos ter várias rotas distintas satisfazendo (4.9). Assim, a restrição (4.10) assegura que, se $y_{ij} > 0$ para algum $(v_i, v_j) \in A$, então $z_{ij} = 1$. Logo, como as variáveis x e z estão relacionadas através de (4.11), teremos uma única sub-rota contendo a origem. Desta forma, uma solução viável irá percorrer todos os vértices uma única vez atendendo respectivamente as restrições de cardinalidade e comprimento.

4.1.2 PCVBP Simétrico

No modelo simétrico, para todas as variáveis x_{ij}^k e z_{ij} considere sempre que $i < j$. Considere ainda w_j^k uma variável binária auxiliar utilizada na formulação. A variável $w_j^k = 1$, indica que a cor k está associada ao vértice v_j e $w_j^k = 0$ caso contrário.

$$min \sum_{i<j} d_{ij} z_{ij} \tag{4.13}$$

sujeito a:

$$\sum_{k \in \phi} w_j^k = 1, \forall v_j \in B \tag{4.14}$$

$$\sum_{i<j} x_{ij}^k + \sum_{j<l} x_{jl}^k = 2w_j^k, \forall v_j \in B, \forall k \in \phi \tag{4.15}$$

$$\sum_{k \in \phi} w_j^k = 2, \forall v_j \in P \tag{4.16}$$

$$\sum_{i<j} x_{ij}^k + \sum_{j<l} x_{jl}^k = w_j^k, \forall v_j \in P, \forall k \in \phi \tag{4.17}$$

$$\sum_{i<j} d_{ij} x_{ij}^k \leq L, \forall k \in \phi \tag{4.18}$$

$$\sum_{i<j} x_{ij}^k \leq Q + 1, \forall k \in \phi \tag{4.19}$$

$$\sum_{v_i, v_j \in S,\ i<j} z_{ij} \leq |S| - 1, \forall S \subset V, |S| \geq 2 \tag{4.20}$$

$$\sum_{k \in \phi} x_{ij}^k = z_{ij}, \forall (v_i, v_j) \in E \ onde \ i < j \tag{4.21}$$

$$x_{ij}^k, z_{ij}, w_j^k \in \{0, 1\}, \forall (v_i, v_j) \in E \ onde \ i < j, \forall k \in \phi \tag{4.22}$$

A formulação matemática do $PCVBP$ simétrico se difere da formulação do $PCVBP$ assimétrico pelas restrições (4.14) ,(4.15) ,(4.16) ,(4.17) e (4.20). Para o modelo simétrico

o conjunto de restrições (4.14) garante que para todo vértice branco v_j, está associado a uma única cor k. O conjunto de restrições (4.15) complementa o conjunto de restrições (4.14) fazendo com que apenas um par de arestas de cor k incida sobre o vértice v_j. O conjunto de restrições (4.16) garante que todo vértices preto v_j está associado a duas cores distintas, o conjunto de restrições (4.17) garante que apenas uma aresta de cor k incida sobre o vértice v_j. O conjunto de restrições (4.20) define as restrições de eliminação sub-rotas. Note neste caso, ao contrário do caso assimétrico, que temos aqui um número exponencial de restrições.

4.2 Heurísticas Propostas

Nesta seção são apresentadas inicialmente três heurística de construção e quatro estruturas de vizinhança para o $PCVBP$. As heurísticas de construção propostas, as heurísticas propostas em [1] e as buscas locais são utilizadas pelas metaheurísticas $GRASP$, VNS e VND na tentativa de se obter melhores resultados aproximados para o $PCVBP$.

Inicialmente descrevemos os métodos propostos para a construção de soluções iniciais e as buscas locais. Numa etapa seguinte, descrevemos a utilização das metaheurísticas $GRASP$, VNS e VND.

Para os algoritmos descritos nas próximas seções considere: s, s' e s^* como sendo soluções viáveis ou inviáveis para o $PCVBP$; v, v_p, v_b e $ultimo\_v\_i$ são variáveis que armazenam respectivamente um vértice qualquer, um vértice preto, um vértice branco e último vértice inserido à solução corrente. Considere ainda:
$Remover\_vertice(lista\_vertices, v)$ um procedimento que remove o vértice v de uma lista de vértices ($lista\_vertices$), $custo\_adicionar(v)$ uma função que retorna o custo de adicionar o vértice v à solução corrente, $Buscar\_vertice(ultimo\_v\_i, lista\_vertices)$ uma função que retorna o vértice pertencente a lista de vértice ($lista\_vertices$) mais próximo do $ultimo\_v\_i$ e caso a lista de vértices esteja vazia retorna -1 e
$Inserir\_vertice(lista\_vertices, v)$ é um procedimento que insere um vértice v a uma lista de vértices ($lista\_vertices$).

4.2.1 Heurísticas de Construção

4.2.1.1 Heurística A1

Este é um algoritmo de construção inspirado no método de *Pétalas*, bastante explorado na resolução do *Problema de Roteamento de Veículos - PVR* [25].

Como os vértices associados ao problema estão localizados no espaço R^2. A heurística $A1$ é um método espacial que consiste em realizar uma varredura circular em todo plano até que se complete 360 graus. Esta varredura é iniciada a partir de um ponto $origem = (x_0, y_0)$ escolhido aleatoriamente como origem dos eixos x e y. Para cada problema a ser resolvido a origem é sorteada dentro do quadrante no plano cartesiano onde os os vértices estão localizados..

Em nossa implementação primeiramente efetuamos a rotação no sentido anti-horário e posteriormente no sentido horário, logo duas soluções viáveis poderão ser geradas.

Inicialmente é escolhido para compor a solução o vértice preto que possui menor ângulo θ (*Figura 4.1*) formado com o eixo x. Seguindo uma rotação no sentido anti-horário (e posteriormente sentido horário) vértices brancos são inseridos um a um à solução corrente levando em consideração seu ângulo. Assim que a restrição de cardinalidade ou comprimento for violada, retorne ao vértice anterior e um vértice preto deve ser inserido de forma que a cadeia (iniciada e terminada em um vértice preto) respeite as restrições de cardinalidade e comprimento. O vértice preto a ser inserido no final da cadeia é aquele que possui o menor ângulo levando em consideração os vértices pretos que não pertencem à solução corrente. Após a inserção do vértice preto uma nova cadeia é iniciada.

Em $A1$, sempre que dois ou mais vértices possuem o mesmo ângulo formado com o eixo x, o vértice escolhido para ser incorporado à solução corrente é aquele que possui a menor distância em relação ao último vértice inserido.

A heurística $A1$ é finalizada quando não existir mais vértices a serem inseridos à solução corrente ou quando uma nova cadeia não pode ser inicializada por não ser possível encontrar um vértice preto que satisfaça as restrições de cardinalidade e comprimento. Neste último caso, $A1$ não é capaz de gerar uma solução inicial viável.

A heurística de construção $A1$ é descrita pelo *Algoritmo 11* e ilustrada pela *Figura 4.2*.

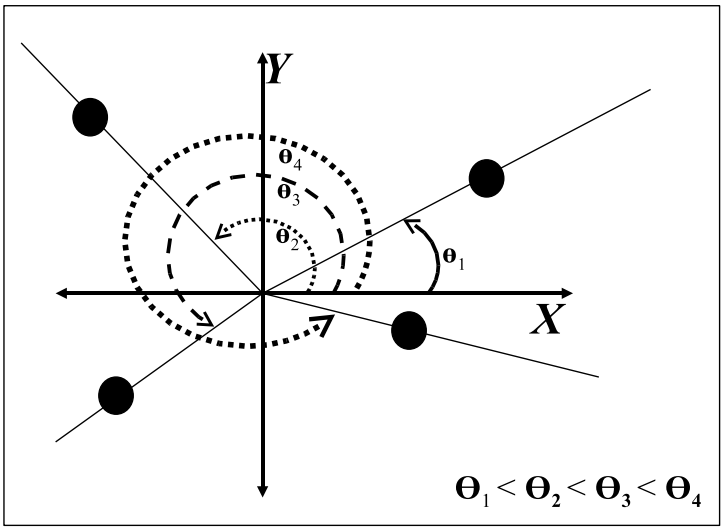


Figura 4.1: Ilustração de como o ângulo θ dever ser considerado.

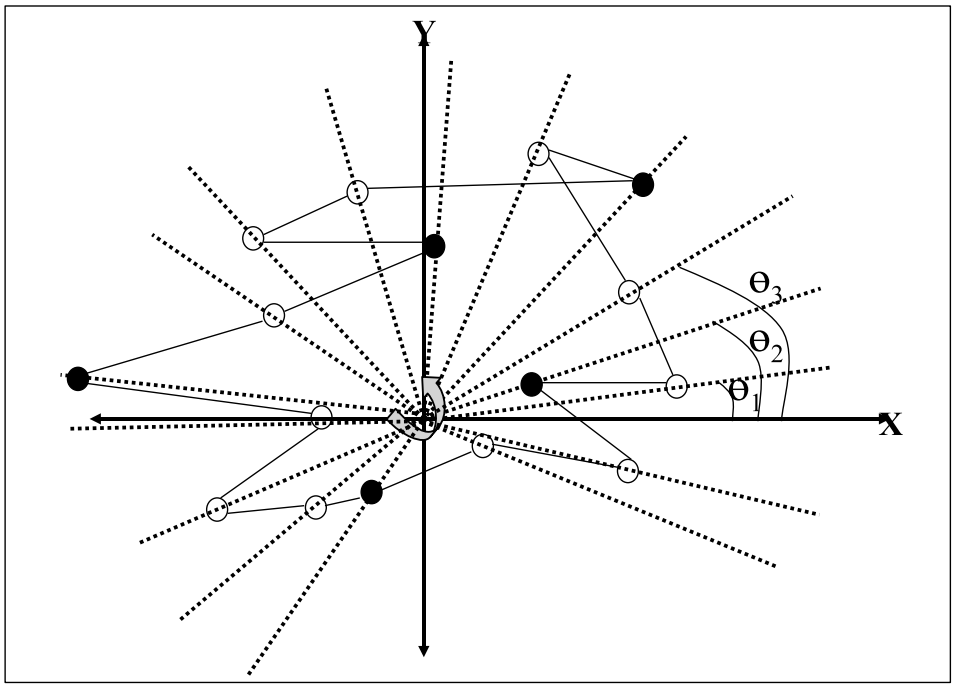


Figura 4.2: Heurística de construção $A1$ (rotação no sentido anti-horário).

Algoritmo 11 A1

1: $s \leftarrow \varnothing$;
2: $s' \leftarrow \varnothing$;
3: $origem \leftarrow Sortear\_Origem$;
4: Tomando como referencia a origem, gerar uma lista de vértices brancos e uma lista de vértices pretos ordenados de forma crescente dos ângulos (θ) para o sentido anti-horário (sah) e de forma decrescente para o sentido horário (sh);
*/ gerar solução s no sentido anti-horário */
5: $v_p \leftarrow$ primeiro elemento da lista de vértices pretos do sentindo anti-horário;
6: $s \leftarrow s \cup \{v_p\}$;
7: $Remover\_vertice(lista\_vertices\_pretos\_sah, v_p)$;
8: $cardinalidade \leftarrow 0$; $comprimento \leftarrow 0$;
9: **para** $i \leftarrow 1$ até $total\_de\_vertices$ **faça**
10: $v_b \leftarrow Buscar\_vertice(ultimo\_v\_i, lista\_vertices\_brancos\_sah)$;
11: **se** $(v_b \neq -1)$ **então**
12: **se** $((cardinalidade + 1 \leq Q) e (comprimento + custo\_adicionar(v_b) \leq L))$ **então**
13: $s \leftarrow s \cup \{v_b\}$;
14: $Remover\_vertice(lista\_vertices\_brancos\_sah, v_b)$;
15: **senão**
16: $s \leftarrow Inserir\_vertice\_preto(s, comprimento, lista\_vertices\_pretos\_sah, lista\_vertices\_brancos\_sah)$;
17: **se** $(s = \varnothing)$ **então**
18: aborte a geração da solução no sentido horário;
19: **senão**
20: $cardinalidade \leftarrow 0$; $comprimento \leftarrow 0$;
21: **fim se**
22: **fim se**
23: **senão**
24: $s \leftarrow Inserir\_vertice\_preto(s, comprimento, lista\_vertices\_pretos\_sah, lista\_vertices\_brancos\_sah)$;
25: **se** $(s = \varnothing)$ **então**
26: aborte a geração da solução no sentido horário;
27: **senão**
28: $cardinalidade \leftarrow 0$; $comprimento \leftarrow 0$;
29: **fim se**
30: **fim se**
31: **fim para**
*/ fim gerar solução s no sentido anti-horário */
*/ gerar solução s' no sentido horário */
.....
*/ fim gerar solução s' no sentido horário */
32: **retorne** s, s';

Algoritmo 12 Inserir_vertice_preto(s, comprimento,lista_vp, lista_vb)

```
1: v_p ← Buscar_vertice(ultimo_v_i, lista_vp);
2: se (v_p = −1) então
3:    s ← ∅;
4:    retorne s;
5: senão
6:    se (coprimento + custo_adicionar(v_p) ≤ L) então
7:       s ← s ∪ {vp};
8:       Remover_vertice(lista_vp, v_p);
9:    senão
10:      inserido ← falso;
11:      enquanto (inserido = falso)e(ultimo_v_i não for um vértice preto) faça
12:         v ← ultimo_v_i;
13:         s ← s − {v};
14:         Inserir_vertice(lista_vb, v);
15:         v_p ← Buscar_vertice(ultimo_v_i, lista_vp);
16:         se (v_p ≠ −1) então
17:            se (comprimento + custo_de_adicionar(v_p)) ≤ L) então
18:               s ← s ∪ {v_p};
19:               Remover_vertice(lista_vp, v_p);
20:               retorne s;
21:            fim se
22:         fim se
23:      fim enquanto
24:      se (inserido=falso) então
25:         s ← ∅;
26:         retorne s;
27:      fim se
28:   fim se
29: fim se
```

4.2.1.2 Heurística Randomizada A1_R

A heurística $A1\_R$ é uma versão *randomizada* de $A1$, onde na fase de escolha do próximo vértice que irá fazer parte da solução corrente é criada uma *Lista Restrita de Candidatos* (LCR). Para compor LCR são escolhidos todos vértices que seu ângulo θ formado com o eixo x, pertencem ao intervalo $\theta_{min} \leq \theta \leq (\theta_{min} + \alpha(\theta_{max} - \theta_{min}))$, onde α é o parâmetro que controla o quanto é gulosa a fase construção e $\alpha \in [0, 1]$, θ_{min} e θ_{max} são respectivamente o ângulo mínimo e ângulo máximo. Os ângulos devem ser considerados como em $A1$ (*veja 4.1*).

Depois de construída a LCR, um elemento é escolhido aleatoriamente para fazer parte da solução corrente.

4.2.1.3 Heurística Randomizada C3_R

Este método incorpora os conceitos propostos na heurística $C3$ [1] apresentada no *Capítulo 2, seção 2.1.1.3*, sendo modificada de forma a torná-la mais adaptativa e randômica.

O principal aspecto que diferencia estas duas heurísticas de construção é que a inserção de um vértice preto à solução corrente se dá pela escolha aleatória de um elemento pertencente a LCR e não pelo critério guloso como em $C3$.

Em relação a LCR assuma que c_{min} representa o menor custo de se inserir um vértice $v \in P$ à solução corrente entre todos os vértices candidatos, c_{max} o maior custo e α o parâmetro que controla o quanto é gulosa a fase construção, $\alpha \in [0, 1]$. A LCR é construída segundo uma função de custo $f(v)$ que retorna o custo c de se inserir v a solução corrente. Se c pertence ao intervalo $c_{min} \leq c \leq c_{min} + \alpha(c_{max} - c_{min})$, v é inserido a LCR.

4.2.2 Heurísticas de Buscas Local

A seguir descrevemos um conjunto de quatro heurísticas de busca local propostas para o $PCVBP$.

4.2.2.1 Busca Local 1 (BL1)

Esta busca local visa "reorganizar"cada cadeia (seqüência de vértices pertencentes a uma solução que inicia e termina com um vértice preto) de forma a minimizar seu custo sem

perder a viabilidade. A $BL1$ funciona da seguinte maneira: para toda cadeia cuja cardinalidade (número de vértices brancos) é maior ou igual a dois, remova cada vértice branco e o re-insira na cadeia corrente no local onde cause o maior decremento do custo da solução sem torná-la inviável.

A Busca Local 1 é descrita pelo *Algoritmo 13*. Para melhor compreensão considere $custo(s)$ uma função que retorna o custo da solução s e $cardinalidade(cadeia^i)$ uma função que retorna a cardinalidade da i-ésima cadeia pertencente solução s.

Algoritmo 13 BL1(s)

1: $s^* \leftarrow s$;
2: $i \leftarrow 0$;
3: $t\_cadeias \leftarrow$ total de cadeias pertencentes a s;
4: **enquanto** $i < t\_cadeias$ **faça**
5: **se** $(cardinalidade(cadeia^i) \geq 2)$ **então**
6: $j \leftarrow 0$;
7: $t\_vertices \leftarrow$ total de vértices brancos pertencentes a $cadeia^i \in s$;
8: **para** $j \leftarrow 0$ até $t\_vertices$ **faça**
9: $v \leftarrow$ j-ésimo vértice branco pertencente a $cadeia^i \in s$;
10: $s' \leftarrow$ Remova $v \in cadeia^i$ e o reinsira na $cadeia^i$ de forma a causar o maior decremento do custo da solução sem torna-la inviável;
11: **se** $(custo(s') < custo(s^*))$ **então**
12: $s^* \leftarrow s'$;
13: **fim se**
14: **fim para**
15: **fim se**
16: $i \leftarrow i + 1$;
17: **fim enquanto**
18: **retorne** s^*;

4.2.2.2 Busca Local 2 (BL2)

A Busca Local 2 ($BL2$) é uma adaptação da fase de refinamento US, pertencente à heurística $GENIUS$ [2] descrita no *Capitulo 2, seção 2.1.1.* A Busca Local 2 consiste em percorrer uma solução viável removendo cada vértice $v_i \in V$ e o re-inserindo usando o procedimento $GENI$ [2], porem a re-inserção do vértice v_i à solução corrente, deve obedecer as restrições de cardinalidade e comprimento. A $BL2$ é descrita pelo *Algoritmo 14*.

Para o *Algoritmo 14* considere n o número de vértices e $solucao\_viavel(s)$ uma função que retorna verdadeiro caso a solução s respeite as restrições de cardinalidade e comprimento, caso contrário retorna falso.

Algoritmo 14 BL2(s)

1: $s^* \leftarrow s$;
2: $s' \leftarrow s$;
3: $v_0, v_1, v_2, ..., v_n \in V$;
4: $i \leftarrow 0$;
5: **enquanto** $(i \leq n)$ **faça**
6: Aplique o procedimento de remoção e inserção considerando os dois possíveis tipos e as duas possíveis orientações para o vértice v_i pertencente a solução s.
$s' \leftarrow$ resultado da remoção e reinserção.
7: **se** $((custo(s') < custo(s^*))$**e**$(solucao\_viavel(s')))$ **então**
8: $s^* \leftarrow s'$;
9: $s \leftarrow s^*$;
10: $i \leftarrow 0$;
11: **senão**
12: $i = i + 1$;
13: **fim se**
14: **fim enquanto**
15: **retorne** s^*;

4.2.2.3 Busca Local 3 (BL3)

A Busca Local 3 ($BL3$) efetua todas as trocas possíveis entre dois vértices distintos (brancos ou pretos) mesmo pertencendo a cadeias diferentes e sempre checando sua viabilidade.

Algoritmo 15 BL3(s)

1: $s' \leftarrow s$;
2: $s^* \leftarrow s$;
3: $v_0, v_1, v_2, ..., v_n \in V$;
4: **para** $i \leftarrow 0$ até n **faça**
5: **para** $j \leftarrow i + 1$ até n **faça**
6: $s' \leftarrow Trocar\_de\_posicao(v_i, v_j, s')$;
7: **se** $((custo(s') < custo(s^*))$**e**$(solucao\_viavel(s')))$ **então**
8: $s* \leftarrow s'$;
9: **fim se**
10: */ desvazer a troca */*
11: $s' \leftarrow Trocar\_de\_posicao(v_i, v_j, s')$;
12: **fim para**
13: **fim para**
14: **retorne** s^*;

No *Algoritmo 15* que descreve a Busca Local 3, $Trocar\_de\_posicao(v_i, v_j, s)$ é uma função que efetua a troca de posição entre os vértices v_i e v_j na solução s.

4.2.2.4 Busca Local 4 (BL4)

A $BL4$ é uma variação da busca local proposta em [1] descrita no *Capítulo 2, seção 2.1.3*. Estas se diferenciam pelo fato de que para cada movimento de troca de arestas que implica na melhoria do custo da solução (sem perda de viabilidade), $BL4$ utiliza esta solução como uma nova semente. Para cada nova semente uma lista de possíveis trocas de arestas é gerada e testada. O critério de parada ocorre quando todas as possíveis trocas de arestas são esgotadas e não há melhoria da semente atual. Veja o *Algoritmo 16*.

Algoritmo 16 BL4(s)

```
1: s* ← s;
2: Q ← lista de todos os subconjuntos de 2 arestas não adjacentes de s;
3: enquanto Q > 0 faça
4:    Z ← subconjunto formado por um elemento de Q;
5:    Remover as arestas ∈ Z de s gerando s';
6:    Reconstruir s';
7:    se (custo(s') < custo(s*))e(solucao_viavel(s')) então
8:       s* ← s';
9:       s ← s';
10:      Q ← lista de todos os subconjuntos de 2 arestas não adjacentes de s;
11:   senão
12:      Q ← Q − Z;
13:   fim se
14: fim enquanto
15: retorne s*;
```

4.2.3 Metaheurísticas

Nesta seção propomos combinações entre os procedimentos construtivos e as buscas locais descritas na seção anterior para serem inseridas na estrutura do $GRASP$ e VNS.

A *Tabela 4.1* ilustra cada versão proposta, onde a coluna **ID** identifica o procedimento implementado, a coluna **Metaheurística** identifica a ou as metaheurísticas utilizadas, a coluna **Construtivo** identifica a heurística utilizada na construção de uma solução inicial e a coluna **Busca Local** representa as heurísticas de refinamento utilizadas.

O procedimento VND' (*Algoritmo 17*) se diferencia da metaheurística VND pelo fato de que é acrescido a este uma *perturbação* da solução quando as heurísticas de busca local não são capazes de melhorar a solução corrente. A idéia contida neste procedimento de perturbação (*Algoritmo 18*) é fazer com que através de movimentos de trocas aleatórias de posições de vértices distintos pertencentes à solução (mesmo sendo movimentos de piora),

ID	**Metaheurística**	**Construtivo**	**Busca Local**
GRASP_1	GRASP	C3_R+F	2-OPT
GRASP_2	GRASP	A1_R+F	2-OPT
GRASP_3	GRASP+VND	C3_R+F	BL1, BL2, BL3, 2-OPT, BL4
GRASP_4	GRASP+VND	A1_R+F	BL1, BL2, BL3, 2-OPT, BL4
GRASP_5	GRASP+VND'+VND	C3_R+F	BL1, BL2, BL3, 2-OPT, BL4
GRASP_6	GRASP+VND'+VND	A1_R+F	BL1, BL2, BL3, 2-OPT, BL4
VNS_1	VNS	C1+F	BL1, BL2, BL3, 2-OPT, BL4
VNS_2	VNS	C2+F	BL1, BL2, BL3, 2-OPT, BL4
VNS_3	VNS	C3+F	BL1, BL2, BL3, 2-OPT, BL4
VNS_4	VNS	A1+F	BL1, BL2, BL3, 2-OPT, BL4

Tabela 4.1: Propostas

seja gerada uma nova solução viável em que as buscas locais possam explorar uma nova vizinhança, permitindo assim sair de ótimos locais ainda distantes de um ótimo global.

Para o *Algoritmo 17* considere NT uma constante que indica o número de trocas a serem efetuadas no procedimento perturba (*Algoritmo 18*). Para o *Algoritmo 18* considere L como sendo uma lista de vértices pertencente à solução corrente.

Algoritmo 17 VND'(s)

1: $s^* \leftarrow s$;
2: $k \leftarrow 1$;
3: **enquanto** $(k \leq 6)$ **faça**
4: $\quad 1 : s' \leftarrow BL1(s^*)$;
5: $\quad 2 : s' \leftarrow BL2(s^*)$;
6: $\quad 3 : s' \leftarrow BL3(s^*)$;
7: $\quad 4 : s' \leftarrow 2 - OPT(s^*)$;
8: $\quad 5 : s' \leftarrow BL4(s^*)$;
9: $\quad 6 : s' \leftarrow Perturba(s^*, NT)$;
$\quad s' \leftarrow VND(s')$;
10: $\quad$ **se** $(custo(s') < custo(s^*))$ **então**
11: $\quad\quad s^* \leftarrow s'$;
12: $\quad\quad k \leftarrow 1$;
13: $\quad$ **senão**
14: $\quad\quad k \leftarrow k + 1$;
15: $\quad$ **fim se**
16: **fim enquanto**
17: **retorne** s^*;

Algoritmo 18 Perturba(s, NT)

1: $L \leftarrow v_0, v_1, ..., v_n / \forall v \in V$;
2: $i \leftarrow 0$;
3: **enquanto** $(i < NT)$ **faça**
4: $troca\_efetuada \leftarrow falso$;
5: **enquanto** $(troca\_efetuada = falso)$ **faça**
6: Escolher dois vértices aleatoriamente a e b pertencentes a L;
7: $s \leftarrow Trocar\_de\_posicao(a, b, s)$;
8: **se** $((solucao\_viavel(s) = verdadeiro)$ **então**
9: $troca\_efetuada \leftarrow verdadeiro$;
10: $L \leftarrow L - \{a, b\}$;
11: $i \leftarrow i + 1$;
12: **senão**
13: $s \leftarrow Trocar\_de\_posicao(a, b, s)$;
14: **fim se**
15: **fim enquanto**
16: **fim enquanto**
17: **retorne** s;

A seguir o *Algoritmo 19* descreve as propostas $GRASP\_3$ e $GRASP\_4$. O *Algoritmo 20* descreve as propostas $GRASP\_5$ e $GRASP\_6$. As versões $GRASP\_1$ e $GRASP\_2$ apresentam a estrutura básica da metaheurística $GRASP$ (veja *Capítulo 3*, *Algoritmo 4*) e as versões VNS_1, VNS_2, VNS_3 e VNS_4 apresentam a estrutura básica do VNS (veja *Capítulo 3*, *Algoritmo 10*).

Algoritmo 19 GRASP_VND (MAX_iterações)

1: $s^* \leftarrow \varnothing$;
2: $custo(s^*) \leftarrow \infty$;
3: $k \longleftarrow 0$;
4: **enquanto** $k \leq$ MAX_iterações **faça**
5: $s \leftarrow$ Construção_GRASP;
6: $s \leftarrow$ VND(s);
7: **se** $custo(s^*) > custo(s)$ **então**
8: $s^* \leftarrow s$;
9: **fim se**
10: $k \longleftarrow k + 1$;
11: **fim enquanto**
12: **retorne** s^*;

Algoritmo 20 GRASP_VND' (MAX_iterações)

1: $s^* \leftarrow \varnothing$;
2: $custo(s^*) \leftarrow \infty$;
3: $k \longleftarrow 0$;
4: **enquanto** $k \leq$ MAX_iterações **faça**
5: $s \leftarrow$ Construção_GRASP;
6: $s \leftarrow$ VND'(s);
7: **se** $custo(s^*) > custo(s)$ **então**
8: $s^* \leftarrow s$;
9: **fim se**
10: $k \longleftarrow k + 1$;
11: **fim enquanto**
12: **retorne** s^*;

Capítulo 5

Detalhes de Implementação

Neste Capítulo serão apresentados alguns pontos aos quais julgamos importantes em nossa implementação. Inicialmente é apresentada a estrutura de dados e a representação da solução para o $PCVBP$, posteriormente é apresentada uma etapa de pré-processamento onde são aplicadas as regras de redução de arestas propostas com o objetivo de reduzir o espaço de busca.

5.1 Estrutura de Armazenamento

É sabida a importância de se escolher uma estrutura de dados adequada para o armazenamento das informações referentes a um problema, principalmente se estamos lidando com estruturas de elevadas dimensões e complexas.

Como as heurísticas e metaheurísticas propostas trabalham com a busca das melhores arestas e/ou vértices para compor uma solução é importante termos um mecanismo eficiente de armazenamento e busca. Para isto, é proposta uma estrutura de armazenamento que permite que seus elementos sejam acessados de forma direta, evitando assim que toda a estrutura ou parte dela seja percorrida para que seja encontrado um determinado elemento.

Para armazenar os vértices é usado um vetor dinâmico de registros (*Figura 5.1*) , onde *id* identifica o vértice, x e y são as coordenadas cartesianas do vértice, *cor* indica se um vértice é preto ou branco e *visitado* se este vértice já foi incorporado a uma solução corrente.

A *Figura 5.2* mostra uma estrutura auxiliar de armazenamento composta por dois vetores (vetor de índice e vetor de arestas) onde o vetor de índices tem como objetivo

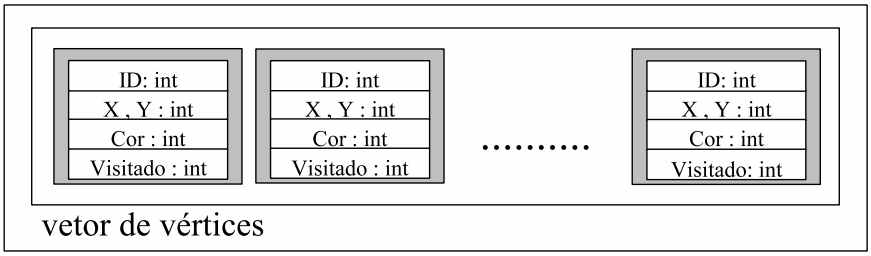


Figura 5.1: Vetor de Vértices

indexar o vetor de arestas. Cada elemento do vetor de índice armazena um índice para um registro do vetor de aresta. Este por sua vez, contem informações como v_i, v_j e o seu custo c_{ij}. Os índices para os registros do vetor de aresta são obtidos da seguinte forma:

$$Indice_{(i,j)} = i\,(n - (i+1)\,/2) + j - i - 1,\ \forall i < j$$

$$onde = \begin{cases} i = \text{id de } v_i \\ j = \text{id de } v_j \\ \text{n} = \text{número de vértices} \end{cases}$$

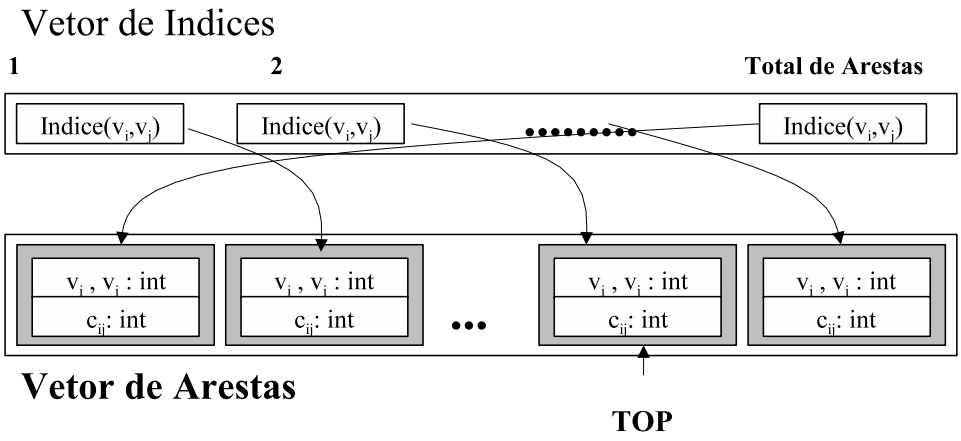


Figura 5.2: Estrutura de Armazenamento

Quando acessamos o elemento $vetor\_de\_indices\left[Indice_{(i,j)}\right]$ estamos na verdade acessando o endereço em que se encontra a $aresta\,(v_i, v_j)$ no vetor de arestas. Logo, se desejamos saber o custo da aresta pertencente ao vértice v_i e v_j basta fazermos:

$$vetor\_de\_arestas\left[vetor\_de\_indices\left[Indice_{(i,j)}\right]\right].c_{ij}$$

Nesta estrutura auxiliar, temos ainda uma variável chamada *TOP*. Esta está diretamente relacionada a etapa de pré-processamento onde são aplicadas as *regras de redução de arestas*. As regras de redução arestas buscam reduzir o espaço de busca através da eliminação de arestas que inviabilizam uma solução. A função desta variável *TOP* é ser

um delimitador entre as arestas que sabidamente não podem fazer parte de uma solução viável e as demais arestas. Assim, a esquerda de *TOP* estão as arestas que devem ser consideradas para o problema e a sua direita estão as arestas descartadas.

5.2 Representação da Solução

Na literatura, para representar uma solução para o *PCV* normalmente é usado um vetor de inteiros, onde cada elemento deste vetor representa um vértice e o índice representa a ordem com que estes são visitados (*Figura 5.3*-a).

Esta mesma representação pode ser usada para o *PCVBP*, porém julgamos não ser a mais indicada, um vez que é muito importante sabermos sempre quais são os sucessores e predecessores de um determinado vértice na solução. Se usarmos a representação habitual sempre que desejarmos obter tais informações seria necessário percorrer o vetor até a posição desejada. Para resolver este problema foi proposta a seguinte representação para uma solução (*Figura 5.3*-b):

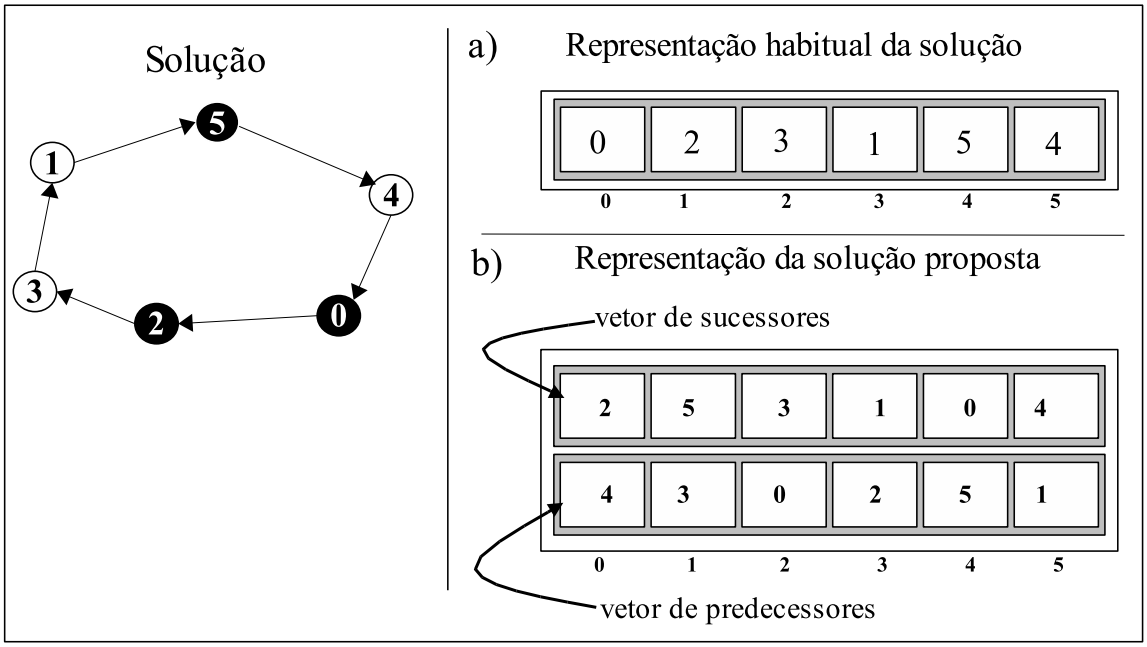


Figura 5.3: Reprentação da Solução

Na *Figura 5.3-b* uma solução é representada por um registro que possui dois vetores de inteiros para armazenar os vértices predecessores e sucessores respectivamente. Para esta representação, o índice representa o vértice e o conteúdo do índice no vetor apresenta o sucessor ou predecessor. Desta forma podemos encontrar os vértices adjacentes de forma direta.

5.3 Regras de redução das arestas

Neste trabalho, propomos algumas regras de redução de arestas no sentido de tentar diminuir o espaço de busca da solução, eliminando implicitamente soluções que sabidamente não farão parte do conjunto de soluções ótimas. Estas regras, normalmente são baseadas em propriedades matemáticas simples.

A seguir são descritas as regras de redução de arestas propostas para o $PCVBP$:

5.3.0.1 Regra 1

Sejam v_j, $v_k \in P$ e v_j o vértice preto mais próximo de $v_i \in B$. Alem disso considere $a_1 = (v_i, v_j)$ e $a_2 = (v_i, v_k)$, tais que se $c_{a_1} + c_{a_2} > L$ então se elimina $a_2 \in A$ (*Figura 5.4*).

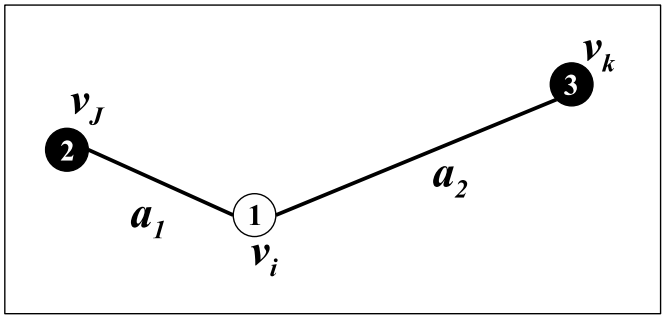


Figura 5.4: Regra de Redução 1

5.3.0.2 Regra 2

Seja v_j o vértice preto mais próximo de $v_i \in B$, v_k o vértice preto mais próximo de $v_l \in B$. Além disso, considere $a_1 = (v_i, v_j)$, $a_2 = (v_i, v_l)$ e $a_3 = (v_l, v_k)$. Se $c_{a_1} + c_{a_2} + c_{a_3} > L$, então se elimina $a_2 \in A$ (*Figura 5.5*).

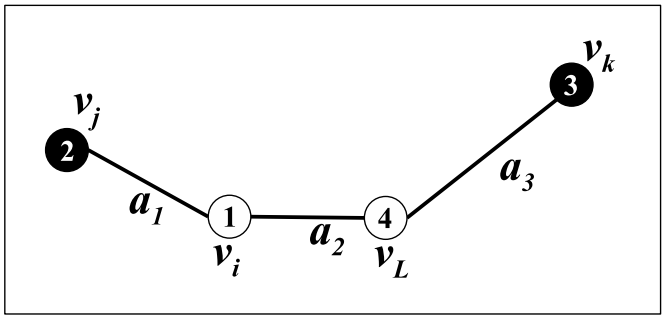


Figura 5.5: Regra de Redução 2

5.3.0.3 Regra 3

Se $Q = 1$ e o total de vértices pretos for igual ao total de vértices brancos eliminar todas as arestas entre vértices pretos.

5.3.0.4 Regra 4

Eliminar todas as arestas que possuem comprimento maior que L.

Capítulo 6

Resultados Computacionais

Para fins de comparação, todas as heurísticas propostas (*Tabela 4.1*), além das heurísticas de construção, viabilidade e de busca local propostas em [1], foram implementadas para a análise de desempenho dos mesmos. Os algoritmos foram codificados em Linguagem C, usando o compilador C++ Builder 5.0 e executados em máquina Intel ® Celeron, 2.8 GHz, 256 MB de memória e sistema operacional Microsoft Windows ®. A formulação matemática proposta foi utilizada para a obtenção de uma solução ótima em algumas instâncias de pequeno porte utilizando o software *GLPK* (GNU Linear Programming Kit) versão 4.8. É importante ressaltar que a formulação matemática para o $PCVBP$ simétrico só foi elaborada ao final deste trabalho, por este motivo somente a formulação do $PCVBP$ assimétrico foi utilizada nos testes.

6.1 Geração dos Problemas Testes e Regras de Redução

As instâncias utilizadas foram obtidas por um gerador de instâncias também implementado neste trabalho, uma vez que, de nosso conhecimento só existe o trabalho proposto em [1] para o $PCVBP$ e as instâncias por eles utilizadas não estão disponíveis.

As instâncias utilizadas em nosso trabalho foram geradas como sugeridos em [1]: escolhe-se um quadrante no plano cartesiano, sorteia-se aleatoriamente o número de vértices desejados dentro deste quadrante. Depois destes criados, uma porcentagem dos vértices é classificada como vértices pretos. A escolha dos vértices a serem classificados como pretos se dá aleatoriamente e obedece as seguintes porcentagens 10%, 20% ou 30% do total de vértices. Os valores de Q e L são propostos da seguinte maneira:

- Valores de Q : $Q = \left\lceil \frac{|B|}{|P|} \right\rceil$, $Q = \left\lceil \frac{|B|}{|P|} \right\rceil + 5$, $Q = \left\lceil \frac{|B|}{|P|} \right\rceil + 10$.

- Valores de L : $L = \frac{z\gamma}{|P|}$, onde z é o custo encontrado pela execução do algoritmo $GENIUS$ para a instância levando em consideração valores iniciais de $Q = L = \infty$ e γ assume valores que variam de [1,75 a 7].

A *Tabela 6.1* exibe as instâncias criadas e para melhor as identificarmos, seus nomes (coluna 4 da *Tabela 6.1*) obedecem a seguinte estrutura, [quadrante]-total de vértices_total de vértices pretos; a coluna **Quadrante** representa o quadrante no plano cartesiano onde os vértices foram gerados e as colunas **T. vértices brancos** e **T. vértices pretos** representam respectivamente o total de vértices brancos e pretos.

Quadrante	**T. vértices brancos**	**T. vértices pretos**	**Nome**
0 - 400	3	2	[0-400]-5_2
0 - 400	4	3	[0-400]-7_3
0 - 400	6	4	[0-400]-10_4
0 - 400	14	5	[0-400]-19_4
0 - 300	21	9	[0-300]-30_9
0 - 100	45	5	[0-100]-50_5
20 - 80	80	20	[20-80]-100_20
40 - 60	140	60	[20-80]-200_60
50 - 300	200	50	[50-300]-250_50
50 - 300	210	90	[50-300]-300_90
0 - 400	280	120	[0-400]-400_120

Tabela 6.1: Instâncias geradas.

Nas *Tabelas 6.2 e 6.3*, para cada instância (*Tabela 6.1*) são sugeridos os valores de Q e L. Após fixados os valores de Q e L numa etapa de pré-processamento são aplicadas as regras de redução, na coluna **% de arestas eliminadas** são exibidas a percentagem de arestas eliminadas, na coluna **ID** é exibido um identificador de cada conjunto formado pela instância e valores sugeridos de Q e L, $\boldsymbol{z}$ é o custo encontrado pela execução do algoritmo $GENIUS$ para a instância levando em consideração valores iniciais de $Q = L = \infty$ e $\boldsymbol{\gamma}$ um fator de ajuste que pertence ao intervalo [1,75 a 7]. Para as células assinaladas com * significa que os valores sugeridos para Q e L não obedecem as regras citadas anteriormente e estes foram obtidos através de testes.

ID	Instância	z	γ	Q	L	% de arestas eliminadas
01	[0-400]-5_2	-	-	2*	380*	20
02	[0-400]-7_3	-	-	2*	400*	0
03	[0-400]-10_4	-	-	2*	500*	4,5
04	[0-400]-19_4	-	-	4*	480*	11,69
05	[0-300]-30_9	1367	3,5	8	532	1,15
06	[0-100]-50_5	578	1,75	9	203	0
07	[0-100]-50_5	578	2	14	232	0
08	[0-100]-50_5	578	2,25	19	261	0
09	[20-80]-100_20	484	2,5	9	61	16,03
10	[20-80]-100_20	484	2,5	14	61	16,03
11	[20-80]-200_60	217	4	3	15	28,4
12	[20-80]-200_60	217	4	8	15	28,4

Tabela 6.2: Conjunto de Instâncias 1.

ID	Instância	z	γ	Q	L	% de arestas eliminadas
13	[50-300]-250_50	2920	4	15	234	14,18
14	[50-300]-300_90	3363	6,69	14	250	5,59
15	[0-400]-400_120	5969	7	13	348	15,37

Tabela 6.3: Conjunto de Instâncias 2.

6.2 Testes Realizados

Usando as instâncias descritas na *Seção 6.1* foram realizadas diversas baterias de testes e os resultados obtidos foram comparados entre os algoritmos propostos neste trabalho (*Tabela 4.1*) e os algoritmos propostos em [1].

Para o *Conjunto de Instâncias 1* (*Tabela 6.2*) foram realizadas dez execuções para cada heurística proposta na *Tabela 4.1*. O critério de parada para os testes foi o número máximo de iterações igual a 100.

Para o *Conjunto de Instâncias 2* (*Tabela 6.3*) como são instâncias maiores e conseqüentemente demandam um maior tempo de processamento, só foram efetuadas 3 execuções e o critério de parada foi o número máximo de iterações igual a 50 ou tempo máximo de execução igual a 43200 segundos. Também só foram selecionados para esta bateria de tes-

tes as heurísticas que apresentaram um bom desempenho para o *Conjunto de Instâncias 1*.

A formulação matemática do $PCVBP$ assimétrico foi utilizada para gerar através do software $GLPK$ soluções exatas para o $PCVBP$. O modelo assimétrico foi utilizado uma vez que, o modelo simétrico só foi proposto na fase conclusão deste trabalho. Infelizmente como já era esperado, a obtenção de uma solução ótima com o uso de métodos exatos só foi possível para instâncias de dimensões muito pequenas. O critério de parada para o otimizador $GLPK$ era encontrar a solução ótima ou tempo máximo de execução igual a quatro horas.

Os resultados obtidos através dos testes são expressos através das *Tabelas 6.4 a 6.19*.

A *Tabela 6.4* exibe as instâncias as quais foram submetidas ao otimizador $GLPK$. Para esta tabela considere **ID Instância** o identificador da instância (*Tabela 6.2*), **Tempo(s)** o tempo em segundos para encontrar a solução ótima ou o tempo limite estabelecido, **Custo Solução** o custo da solução encontrada e **Limite Inferior** indica que o custo da solução ótima nunca é inferior a este limite. As células assinaladas com $-$ significa que o software $GLPK$ não conseguiu encontrar uma solução viável ao $PCVBP$ (não convergiu) e para as células assinaladas com $*$ significa a solução encontrada não é a solução ótima.

Para as demais tabelas considere a coluna **Algoritmo** como sendo os algoritmos propostos por este trabalho e os algoritmos propostos em [1], a coluna $\boldsymbol{\alpha}$ indica o quanto é gulosa a fase construção da metaheurística $GRASP$. As colunas **Melhor S.**, **S. Média**, **Pior S.**, **T. Médio(s)** e **Total de S.** representam respectivamente melhor solução encontrada, solução média levando em consideração todas as soluções viáveis geradas, pior solução, tempo médio de execução em segundos e o total de soluções encontradas para cada algoritmo num total de dez execuções. Para a coluna **Melhor S.** o valor $X_{(}y)$ de cada célula indica que a solução X foi encontrada y vezes. As células em que seu conteúdo está em negrito indicam as melhores soluções encontradas em suas respectivas colunas.

ID Instância	*Tempo(s)*	Custo Solução	Limite Inferior
01	1	467	
02	1	768	
03	2	1080	
04	14400	1474*	1053
05	14400	1704*	1197,655172
06	14400	-	503,4285714
07	14400	-	503,4285714
08	14400	726*	503,4285714

Tabela 6.4: Soluções encontradas pelo otimizador $GLPK$ (* significa que a solução viável encontrada não é ótima e - significa que o $GLPK$ em quatro de horas de execução não conseguiu encontrar nenhuma solução viável).

Algoritmo	α	Melhor S.	S. Média	Pior S.	T. Médio(s)	Total de S.
GRASP_1	10%	$\mathbf{467}_{(10)}$	**467**	**467**	0,021	10
GRASP_2	10%	$\mathbf{467}_{(10)}$	**467**	**467**	0,02	10
GRASP_3	10%	$\mathbf{467}_{(10)}$	**467**	**467**	0,055	10
GRASP_4	10%	$\mathbf{467}_{(10)}$	**467**	**467**	0,018	10
GRASP_5	10%	$\mathbf{467}_{(10)}$	**467**	**467**	0,06	10
GRASP_6	10%	$\mathbf{467}_{(10)}$	**467**	**467**	0,033	10
VNS_1		$\mathbf{467}_{(10)}$	**467**	**467**	0,035	10
VNS_2		$\mathbf{467}_{(10)}$	**467**	**467**	0,03	10
VNS_3		$\mathbf{467}_{(10)}$	**467**	**467**	0,017	10
VNS_4		$\mathbf{467}_{(10)}$	**467**	**467**	0,021	10
C1+F+2-OPT [1]		$\mathbf{467}_{(2)}$	530,6	573	0,003	5
C2+F+2-OPT [1]		$\mathbf{467}_{(6)}$	527,5	573	0,003	14
C3+F+2-OPT [1]		$\mathbf{467}_{(6)}$	515,1	573	0,002	11

Tabela 6.5: Instância 01: [0-400]-5_2, total de vértices 5, 2 vértices pretos, Q = 2 e L = 380.

Pelos resultados obtidos na *Tabela 6.4* e *Tabela 6.5*, observa-se que todas as heurísticas implementadas atingiram a solução ótima para a *instância 01*. Para os algoritmos propostos (*Tabela 4.1*) em 100% (10 execuções) das execuções a solução ótima foi encontrada.

Algoritmo	$\boldsymbol{\alpha}$	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	10%	$\mathbf{768}_{(10)}$	**768**	**768**	$0,0325$	10
GRASP_2	10%	$\mathbf{768}_{(10)}$	**768**	**768**	$0,013$	10
GRASP_3	10%	$\mathbf{768}_{(10)}$	**768**	**768**	$0,311$	10
GRASP_4	10%	$\mathbf{768}_{(10)}$	**76**8	**768**	$0,206$	10
GRASP_5	10%	$\mathbf{768}_{(10)}$	**768**	**768**	$0,560$	10
GRASP_6	10%	$\mathbf{768}_{(10)}$	**76**8	**768**	$0,405$	10
VNS_1		$\mathbf{768}_{(10)}$	**768**	**768**	$0,575$	10
VNS_2		–	–	–	$0,026$	10
VNS_3		$\mathbf{768}_{(10)}$	**768**	**768**	$0,174$	10
VNS_4		$\mathbf{768}_{(10)}$	**768**	**76**8	$0,336$	10
C1+F+2-OPT [1]		$\mathbf{768}_{(10)}$	**768**	**768**	$0,006$	10
C2+F+2-OPT [1]		–	–	–	$0,001$	0
C3+F+2-OPT [1]		$\mathbf{768}_{(6)}$	$783,36$	833	$0,002$	11

Tabela 6.6: Instância 02: [0-400]-7_3, total de vértices 7, 3 vértices pretos, Q = 2 e L = 400.

A *Tabela 6.6* mostra que a solução ótima é encontrada pela maioria das heurísticas analisadas, com exceção dos algoritmos VNS_2 e $C2 + F + 2 - OPT$ [1] que não conseguiram encontrar uma solução viável para o $PCVBP$. Nos demais algoritmos propostos a solução ótima é encontrada em 100% das execuções.

Pela descrição da *Table 4.1*, observamos que a única metaheurística proposta que usa a heurística construtiva $C2$ seguido pelo módulo de viabilização F é a versão VNS_2. Portanto ao menos para a *instância 02* o módulo $C2 + F$ parece ter um desempenho pior que os construtivos $C1$ e $C3$. Em termos de características dessa instância, observa-se pela *Tabela 6.2*, que esta faz parte de um conjunto de instâncias onde as regras de redução não conseguiram eliminar nenhuma aresta.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	10%	$1138_{(10)}$	1138	1138	0,100	10
GRASP_2	5%	$\mathbf{1080}_{(5)}$	1083	1086	0,030	10
GRASP_3	20%	$\mathbf{1080}_{(5)}$	1083	1086	1,864	10
GRASP_4	10%	$\mathbf{1080}_{(10)}$	**1080**	**1080**	2,55	10
GRASP_5	10%	$\mathbf{1080}_{(10)}$	**1080**	**1080**	2,54	10
GRASP_6	10%	$\mathbf{1080}_{(10)}$	**1080**	**1080**	3,48	10
VNS_1		$\mathbf{1080}_{(10)}$	**1080**	**1080**	2,83	10
VNS_2		$\mathbf{1080}_{(10)}$	**1080**	**1080**	1,16	10
VNS_3		$\mathbf{1080}_{(10)}$	**1080**	**1080**	1,75	10
VNS_4		$\mathbf{1080}_{(10)}$	**1080**	**1080**	2,43	10
C1+F+2-OPT [1]		$1167_{(3)}$	1188,7	1198	0,08	10
C2+F+2-OPT [1]		$1138_{(1)}$	1269,64	1447	0,001	14
C3+F+2-OPT [1]		$1086_{(1)}$	1257,93	1398	0,0012	16

Tabela 6.7: Instância 03: [0-400]-10_4, total de vértices 10, 4 vértices pretos, Q = 2 e L = 500.

Os resultados da *Tabela 6.7* mostraram que com exceção do $GRASP\_1$, todos os demais algoritmos propostos encontraram a solução ótima para a *instância 03*. Contudo, isso não ocorre com os algoritmos propostos em [1].

Para esta instância as versões $GRASP\_4$, $GRASP\_5$, $GRASP\_6$, VNS_1, VNS_2, VNS_3 e VNS_4 se mostraram mais regulares, encontrando em todas as 10 execuções a solução ótima.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	20%	$1545_{(2)}$	$1548,9$	1558	$0,64$	10
GRASP_2	5%	$1572_{(1)}$	1669	1791	$0,050$	10
GRASP_3	20%	$\mathbf{1455}_{(10)}$	**1455**	**1455**	$3,46$	10
GRASP_4	10%	$\mathbf{1455}_{(10)}$	**1455**	**1455**	$0,96$	10
GRASP_5	10%	$\mathbf{1455}_{(10)}$	**1455**	**1455**	$4,914$	10
GRASP_6	10%	$\mathbf{1455}_{(10)}$	**1455**	**1455**	$1,81$	10
VNS_1		$\mathbf{1455}_{(10)}$	**1455**	**1455**	$4,41$	10
VNS_2		–	–	–	$0,59$	0
VNS_3		$\mathbf{1455}_{(10)}$	**1455**	**1455**	$2,8755$	10
VNS_4		$\mathbf{1455}_{(10)}$	**1455**	**1455**	$2,93$	10
C1+F+2-OPT [1]		$1456_{(1)}$	$1795,83$	1889	$0,021$	6
C2+F+2-OPT [1]		–	–	–	$0,006$	0
C3+F+2-OPT [1]		$1604_{(1)}$	1883	2004	$0,005$	6

Tabela 6.8: Instância 04: [0-400]-19_5, total de vértices 19, 5 vértices pretos, Q = 4 e L = 480.

Os resultados da *Tabela 6.8* mostram que para a *instância 04* as únicas heurísticas que não conseguiram obter soluções de menor custo que a solução encontrada pelo otimizador $GLPK$ foram: $GRASP\_1$, $GRASP\_2$, VNS_2, $C2 + F + 2 - OPT$ [1] e $C3 + F + 2 - OPT$ [1]. Além disso, podemos observar que as heurísticas propostas superaram os resultados obtidos pelas heurísticas propostas em [1] e que a partir da *instância 04* há uma enorme diferença do tempo exigido pelo otmizador $GLPK$ (*Tabela 6.4*)e as heurísticas implementadas. É importante ressaltar que o valor encontrado pelo $GLPK$ (1474) não representa o custo da solução ótima.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	30%	$1551_{(2)}$	1590, 1	1633	3, 51	10
GRASP_2	30%	$1874_{(1)}$	2974, 4	2412	0, 425	10
GRASP_3	30%	$\mathbf{1442}_{(10)}$	**1442**	**1442**	20, 47	10
GRASP_4	30%	$\mathbf{1442}_{(10)}$	**1442**	**1442**	19, 32	10
GRASP_5	30%	$\mathbf{1442}_{(10)}$	**1442**	**1442**	29, 76	10
GRASP_6	30%	$\mathbf{1442}_{(10)}$	**1442**	**1442**	27, 28	10
VNS_1		$\mathbf{1442}_{(10)}$	**1442**	**1442**	20, 43	10
VNS_2		$\mathbf{1442}_{(10)}$	**1442**	**1442**	19, 48	10
VNS_3		$\mathbf{1442}_{(10)}$	**1442**	**1442**	17, 41	10
VNS_4		$\mathbf{1442}_{(10)}$	**1442**	**1442**	18, 23	10
C1+F+2-OPT [1]		$1474_{(1)}$	1705, 2	1754	0, 079	20
C2+F+2-OPT [1]		$1667_{(1)}$	2348, 85	3298	0, 033	20
C3+F+2-OPT [1]		$1530_{(1)}$	1811, 7	2118	0, 033	20

Tabela 6.9: Instância 05: [0-300]-30_9, total de vértices 30, 5 vértices pretos, Q = 8 e L = 304 (z = 1367 e $\gamma = 2$).

Novamente notamos que na *instância 05* as heurísticas implementadas alcançaram soluções de melhor qualidade que o $GLPK$, gastando tempo bem menores. Nesta instância as melhores soluções foram obtidas pelas heurísticas $GRASP\_3$, $GRASP\_4$, $GRASP\_5$, $GRASP\_6$, VNS_1, VNS_2, VNS_3 e VNS_4 que sempre alcançam o mesmo valor em todas as 10 execuções.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	10%	$678_{(10)}$	678	678	23,33	10
GRASP_2	10%	–	–	–	0,050	0
GRASP_3	10%	$\mathbf{639}_{(10)}$	**639**	**639**	134,33	10
GRASP_4	10%	–	–	–	0,0438	0
GRASP_5	10%	$\mathbf{639}_{(6)}$	640,5	642	122,92	10
GRASP_6	10%	–	–	–	0,042	0
VNS_1		$692_{(1)}$	735	761	31,16	5
VNS_2		–	–	–	21,83	0
VNS_3		$642_{(3)}$	649,5	663	78,82	10
VNS_4		–	–	–	0,018	0
C1+F+2-OPT [1]		–	–	–	0.025	10
C2+F+2-OPT [1]		–	–	–	0,025	0
C3+F+2-OPT [1]		$690_{(1)}$	793,5	899	0,023	15

Tabela 6.10: Instância 06: [0-100]-50_5, total de vértices 50, 5 vértices pretos, Q = 9 e L = 203 (z = 578 e γ = 1,75).

A *instância 06* se caracteriza como uma instância "difícil". Isso porque muitas heurísticas ($GRASP\_2$, $GRASP\_4$, $GRASP\_6$, VNS_1, VNS_2 e as heurísticas propostas em [1]) e até mesmo o otimizador $GLPK$ não conseguiram obter uma solução viável para o $PCVBP$.

Esta instância faz parte do conjunto de problemas testes onde as regras de redução não conseguem eliminar nenhuma aresta (veja *Tabela 6.2*).

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	5%	$695_{(1)}$	$711,4$	722	$23,75$	10
GRASP_2	2%	$832_{(1)}$	$870,6$	914	$0,33$	10
GRASP_3	5%	$\mathbf{609}_{(10)}$	**609**	**609**	$98,10$	10
GRASP_4	2%	$\mathbf{609}_{(10)}$	**609**	**609**	$12,34$	10
GRASP_5	5%	$\mathbf{609}_{(10)}$	**609**	**609**	$142,86$	10
GRASP_6	2%	$\mathbf{609}_{(9)}$	$609,4$	613	$14,28$	10
VNS_1		$612_{(10)}$	612	612	$121,42$	10
VNS_2		$\mathbf{609}_{(10)}$	**609**	**609**	$79,70$	10
VNS_3		$\mathbf{609}_{(10)}$	**609**	**609**	$87,04$	10
VNS_4		$\mathbf{609}_{(10)}$	**609**	**609**	$12,99$	10
C1+F+2-OPT [1]		$623_{(4)}$	$683,6$	769	$0,31$	10
C2+F+2-OPT [1]		$656_{(1)}$	$796,9$	939	$0,24$	16
C3+F+2-OPT [1]		$754_{(1)}$	$837,7$	939	$0,23$	17

Tabela 6.11: Instância 07: [0-100]-50_5, total de vértices 50, 5 vértices pretos, Q = 14 e L = 232 (z = 578 e $\gamma = 2$).

Na *instância 07* o $GLPK$ não conseguiu obter nenhuma solução viável para o $PCVBP$ em 4 horas de execução (*Tabela 6.4*), mas ao contrário da *instância 06* (*Tabela 6.10*), todas as heurísticas conseguiram gerar soluções viáveis (*Tabela 6.11*) em poucos segundos. O melhor desempenho foi novamente obtido pelas versões $GRASP\_3$, $GRASP\_4$, $GRASP\_5$, $GRASP\_6$, VNS_2, VNS_3 e VNS_4.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	5%	$676_{(2)}$	680,6	684	23,76	10
GRASP_2	2%	$909_{(1)}$	927,1	962	0,88	10
GRASP_3	5%	$\mathbf{582}_{(10)}$	**582**	**582**	73,72	10
GRASP_4	2%	$\mathbf{582}_{(10)}$	**582**	**582**	26,72	10
GRASP_5	5%	$\mathbf{582}_{(10)}$	**582**	**582**	122,18	10
GRASP_6	2%	$\mathbf{582}_{(10)}$	**582**	**582**	39,45	10
VNS_1		$\mathbf{582}_{(10)}$	**582**	**582**	69,83	10
VNS_2		$\mathbf{582}_{(10)}$	**582**	**582**	81,9	10
VNS_3		$\mathbf{582}_{(10)}$	**582**	**582**	79,17	10
VNS_4		$\mathbf{582}_{(10)}$	**582**	**582**	29,24	10
C1+F+2-OPT [1]		$599_{(2)}$	619,9	633	0,29	10
C2+F+2-OPT [1]		$702_{(1)}$	838,8	1030	0,23	20
C3+F+2-OPT [1]		$687_{(1)}$	746,3	956	0,22	20

Tabela 6.12: Instância 08: [0-100]-50_5, total de vértices 50, 5 vértices pretos, Q = 19 e L = 261 (z = 578 e γ = 2,25).

Para a *instância 08* o $GLPK$ (*Tabela 6.4*) obteve em 4 horas de execução uma solução aproximada (726) muito pior que as heurísticas implementadas (*Tabela 6.12*). Os melhores resultados e as versões mais regulares foram obtidos novamente pelo $GRASP\_3$, $GRASP\_4$, $GRASP\_5$, $GRASP\_6$, VNS_1, VNS_2, VNS_3 e VNS_4.

Algoritmo	$\boldsymbol{\alpha}$	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	5%	$650_{(1)}$	$679,3$	707	$85,10$	10
GRASP_2	2%	–	–	–	$0,08$	0
GRASP_3	5%	$\mathbf{487}_{(1)}$	$495,7$	516	$208,80$	10
GRASP_4	2%	–	–	–	$0,08$	0
GRASP_5	5%	$488_{(1)}$	**491,3**	**492**	$300,02$	10
GRASP_6	2%	–	–	–	$0,09$	0
VNS_1		$\mathbf{487}_{(1)}$	$495,22$	502	$278,65$	9
VNS_2		–	–	–	$0,805$	0
VNS_3		$491_{(2)}$	494	504	$196,26$	10
VNS_4		–	–	–	$0,054$	0
C1+F+2-OPT [1]		$707_{(1)}$	707	707	$0,29$	1
C2+F+2-OPT [1]		–	–	–	$0,23$	0
C3+F+2-OPT [1]		$784_{(1)}$	784	784	$0,22$	1

Tabela 6.13: Instância 09: [20-80]-100_20, total de vértices 100, 20 vértices pretos, Q = 9 e L = 61 (z = 484 e γ = 2,5).

A partir da *instância 09* os resultados comparativos serão somente entre as heurísticas. Para esta instância, várias versões não conseguiram obter soluções viáveis para o $PCVBP$. As melhores soluções foram obtidas velo $GRASP\_3$ e VNS_1 e o melhor desempenho médio foi obtido pelo $GRASP\_5$.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	5%	$650_{(1)}$	664,8	686	94,4	10
GRASP_2	2%	–	–	–	0,08	0
GRASP_3	5%	$\mathbf{483}_{(1)}$	488,9	493	242,79	10
GRASP_4	2%	–	–	–	0,094	0
GRASP_5	5%	$484_{(2)}$	**488,6**	**491**	260,90	10
GRASP_6	2%	–	–	–	0,099	0
VNS_1		$491_{(2)}$	494,5	504	255,94	8
VNS_2		–	–	–	84,16	0
VNS_3		$486_{(1)}$	492,5	499	188,78	10
VNS_4		–	–	–	0,076	0
C1+F+2-OPT [1]		–	–	–	1,95	0
C2+F+2-OPT [1]		–	–	–	0,61	0
C3+F+2-OPT [1]		$663_{(1)}$	708,5	754	1,3	2

Tabela 6.14: Instância 10: [20-80]-100_20, total de vértices 100, 20 vértices pretos, Q = 14 e L = 61 (z = 484 e γ = 2,5).

Resultados similares a instância anterior é verificada na *instância 10*, onde percebe-se novamente a dificuldade de várias heurísticas em obter soluções válidas. Neste caso, a melhor solução foi obtida pelo $GRASP\_3$ e a melhor média pelo $GRASP\_5$.

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	5%	$329_{(1)}$	343, 8	360	245, 16	10
GRASP_2	2%	$620_{(1)}$	648	665	2, 85	4
GRASP_3	5%	$\mathbf{255}_{(2)}$	261, 66	290	948, 88	10
GRASP_4	2%	$263_{(1)}$	271, 75	280	324, 47	4
GRASP_5	5%	$257_{(1)}$	262, 88	273	1139, 54	9
GRASP_6	2%	$263_{(1)}$	269, 33	277	244, 04	3
VNS_1		$256_{(1)}$	**259,2**	**260**	5121, 014	10
VNS_2		–	–	–	195, 95	0
VNS_3		$258_{(1)}$	262, 22	262	811, 16	8
VNS_4		$263_{(1)}$	264, 5	266	114, 30	2
C1+F+2-OPT [1]		$442_{(1)}$	443, 5	445	2, 12	2
C2+F+2-OPT [1]		–	–	–	0, 0007	0
C3+F+2-OPT [1]		–	–	–	2, 25	0

Tabela 6.15: Instância 11: [40-60]-200_60, total de vértices 600, 60 vértices pretos, Q = 3 e L = 15 (z = 217 e $\gamma = 4$).

Algoritmo	α	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_1	5%	$341_{(1)}$	351, 83	373	201, 27	7
GRASP_2	2%	$608_{(1)}$	653, 88	673	7, 017	8
GRASP_3	5%	$223_{(1)}$	224, 8	228	280, 57	4
GRASP_4	2%	$222_{(1)}$	226, 62	230	184, 75	8
GRASP_5	5%	$221_{(1)}$	234, 11	314	422, 03	9
GRASP_6	2%	$\mathbf{220}_{(1)}$	224	227	387, 62	7
VNS_1		$221_{(1)}$	**222,5**	**224**	1964, 44	10
VNS_2		–	–	–	235, 09	0
VNS_3		$222_{(1)}$	224, 65	228	421, 86	8
VNS_4		$224_{(1)}$	226, 57	230	175, 59	7
C1+F+2-OPT [1]		$260_{(1)}$	260	260	10, 14	1
C2+F+2-OPT [1]		–	–	–	3, 54	0
C3+F+2-OPT [1]		$329_{(1)}$	329	329	3, 27	1

Tabela 6.16: Instância 12: [40-60]-200_60, total de vértices 200, 60 vértices pretos, Q = 8 e L = 15 (z = 217 e $\gamma = 4$).

Nas *instâncias 11* e *12* (*Tabelas 6.15* e *6.16*) a melhor solução foi encontrada pelo $GRASP\_3$ e a melhor média pelas versões $GRASP\_5$ e VNS_1.

A partir dos resultados ilustrados nas *Tabelas 6.5* a *6.16*, foram selecionados apenas os algoritmos $GRASP\_3$, $GRASP\_5$ e VNS_1 para realizar os testes com o *Conjunto de Instâncias 2* (*Tabela 6.3*), uma vez que, são instâncias maiores e conseqüentemente possuem um tempo de processamento mais elevado. O algoritmo $GRASP\_3$ foi selecionado por apresentar o maior número de melhores soluções, o algoritmo $GRASP\_5$ por apresentar a melhor média entre as soluções obtidas e o VNS_1 por ser o algoritmo VNS com melhor desempenho. Para estes algoritmos e o *Conjunto de Instâncias 2* foram realizadas apenas três execuções e o critério de parada foi o número máximo de iterações igual a 50 ou o tempo máximo de execução igual a 43200 segundos.

Algoritmo	$\boldsymbol{\alpha}$	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_3	10%	$2949_{(1)}$	2969,3	2989	20894,5	3
GRASP_5	10%	$\mathbf{2903_{(1)}}$	2923,3	2939	22190,3	3
VNS_1		$2912_{(1)}$	**2922**	**2938**	18628	3

Tabela 6.17: Instância 13: [50-300]-250_50, total de vértices 250, 50 vértices pretos, Q = 15 e L = 234 (z = 2920 e $\gamma = 4$).

Algoritmo	$\boldsymbol{\alpha}$	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_3	10%	$3378_{(1)}$	3381,33	**3384**	31560,4	3
GRASP_5	10%	$3362_{(1)}$	**3377,33**	3403	43200	3
VNS_1		$\mathbf{3342_{(1)}}$	3381,3	3443	37032,88	3

Tabela 6.18: Instância 14: [50-300]-300_90, total de vértices 300, 90 vértices pretos, Q = 14 e L = 250 (z = 3363 e γ = 6,69).

Algoritmo	$\boldsymbol{\alpha}$	**Melhor S.**	**S. Média**	**Pior S.**	**T. Médio(s)**	**Total de S.**
GRASP_3	10%	$5978_{(1)}$	5978	**5978**	43200	1
GRASP_5	10%	$5976_{(1)}$	**5977**	5978	43200	2
VNS_1		$\mathbf{5929_{(1)}}$	5985,5	6042	43200	2

Tabela 6.19: Instância 15: [0-400]-400_120, total de vértices 400, 120 vértices pretos, Q = 13 e L = 348 (z = 5969 e $\gamma = 7$).

Os resultados das *Tabelas 6.17* a *6.19* mostram um comportamento semelhante dos

três algoritmos observando uma pequena superioridade do $GRASP\_5$ em termos da solução média.

Fazendo uma análise quantitativa dos dados expostos através das *Tabelas 6.5 a 6.16* (*conjunto de instâncias 1*) se obtem os seguintes gráficos expressos nas *Figuras 6.1, 6.2 e 6.3*, que representam respectivamente o número de vezes em cada heurística obteve a melhor solução, a melhor média entre as soluções viáveis encontradas e as melhores piores soluções. Através destes gráficos podemos constatar que o algoritmo que obteve melhor desempenho quanto a melhor solução foi a metaheurística $GRASP\_3$ e a melhor média foi obtida pela metaheurística $GRASP\_5$.

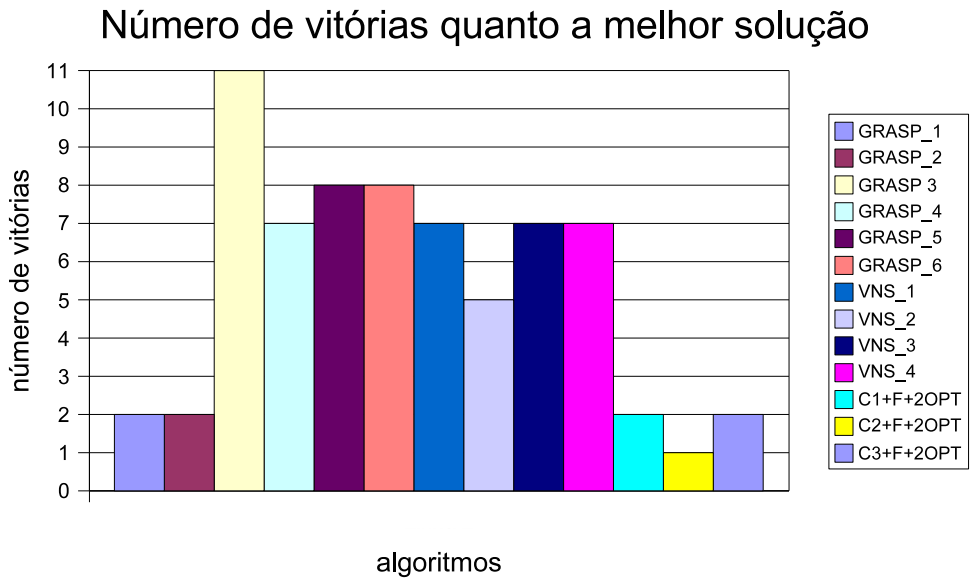


Figura 6.1: Comparação das Heurísticas quanto ao número de melhores soluções obitidas (Conjunto de instâncias 1).

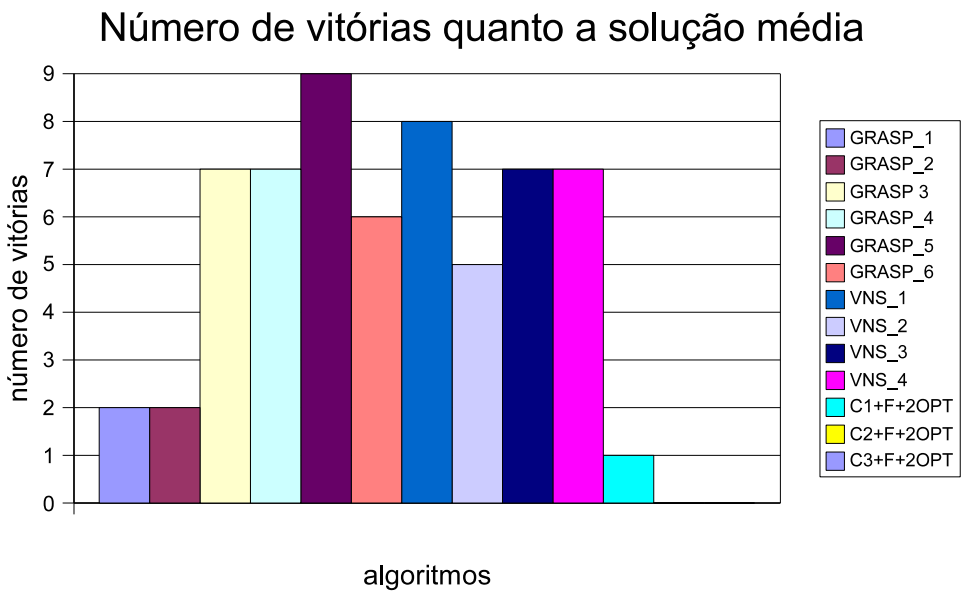


Figura 6.2: Comparação das Heurísticas quanto ao número de melhores soluções médias obitidas (Conjunto de instâncias 1).

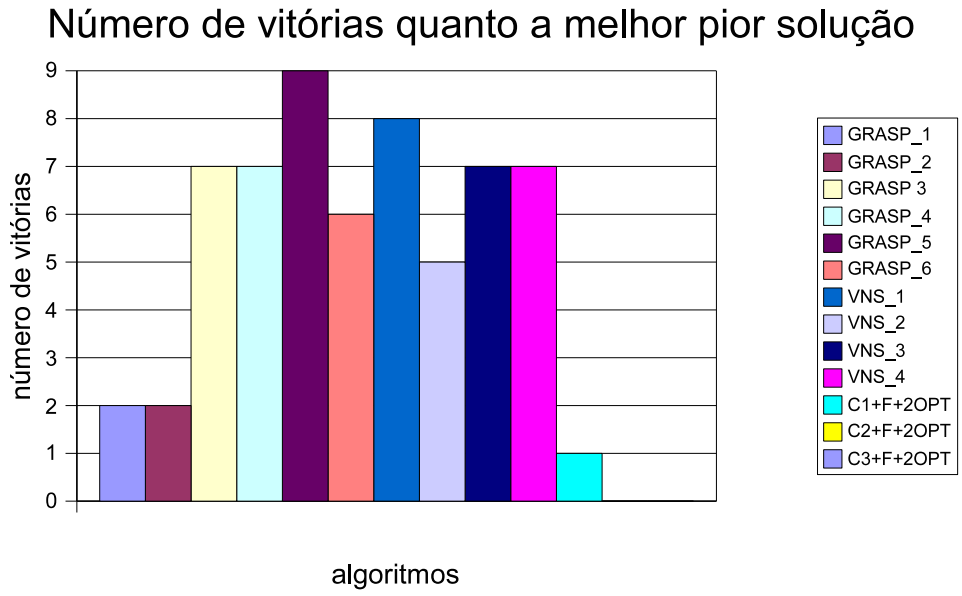


Figura 6.3: Comparação das Heurísticas quanto ao número de melhores piores soluções obitidas (Conjunto de instâncias 1).

Para o *Conjunto de Instâncias 2* (*Tabelas 6.17,6.18 e 6.19*) os melhores resultados foram obtidos pelo VNS_1 e $GRASP\_5$. O VNS_1 apresentou o maior número de melhores soluções e o $GRASP\_5$ apresentou a melhor média.

6.3 Análise Probabilística

Para participar desta análise foram selecionados os algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1 que obtiveram melhor média (veja *Figura 6.2*) entre as soluções viáveis e que apresentam heurísticas de construção distintas.

Neste tipo de experimento, para cada instância selecionada executou-se cada algoritmo 100 vezes usando sementes distintas para a geração de números aleatórios. O critério de parada neste caso é um valor alvo (o algoritmo é encerrado quanto encontra uma solução viável cujo valor é igual ou inferior ao valor alvo) ou um tempo limite.

Em cada execução, o tempo de processamento para atingir a solução alvo é armazenado em ordem crescente. Ao *i-ésimo* tempo de execução ordenado (t_i) é associada uma probabilidade empírica $p_i = \frac{(i-1/2)}{n}$. Os pontos $Z_i = (t_i, p_i)$ para $i = 1, ..., n$ são plotados em um gráfico, segundo as definições dada em [26]. Nestes gráficos, cada curva representa o desempenho de um algoritmo e quanto mais a esquerda estiver a curva, melhor será o desempenho do algoritmo associado.

Para cada instância foram estabelecidos dois alvos baseados nas soluções obtidas nos testes anteriores e um tempo limite de execução:

- **alvo 1:** corresponde a valor médio das soluções médias obtidas pelos algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1.
- **alvo 2:** corresponde ao melhor valor médio obtido pelos algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1.
- **Tempo Limite:** corresponde ao valor médio do tempo de execução dos algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1.

As instâncias escolhidas foram as instâncias 07 e 12 (*Tabela 6.2*), estas foram escolhidas pois, todos os algoritmos selecionados conseguiram gerar soluções viáveis.

Os resultados desta bateria de testes são apresentados nas *Figuras 6.4* a *6.7*. Nos gráficos representados pelas *Figuras 6.4* e *Figuras 6.5* a heurística VNS_C1 não conseguiu atingir o alvo no tempo limite.

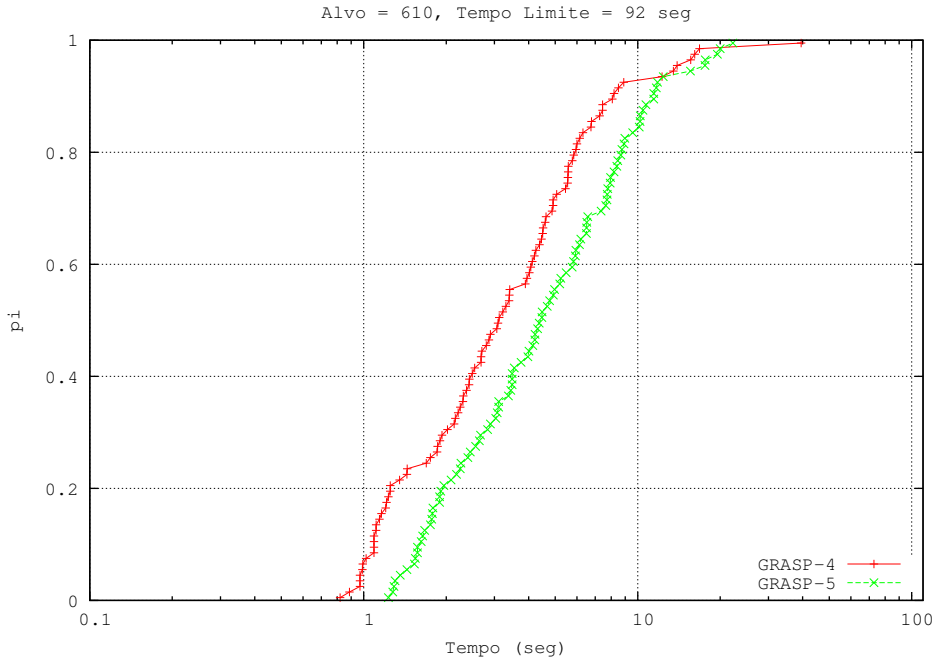


Figura 6.4: Comparação entre os algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1 (não conseguiu atingir a solução alvo) para a instância 7, solução alvo = 610 e tempo limite = 92 segundos.

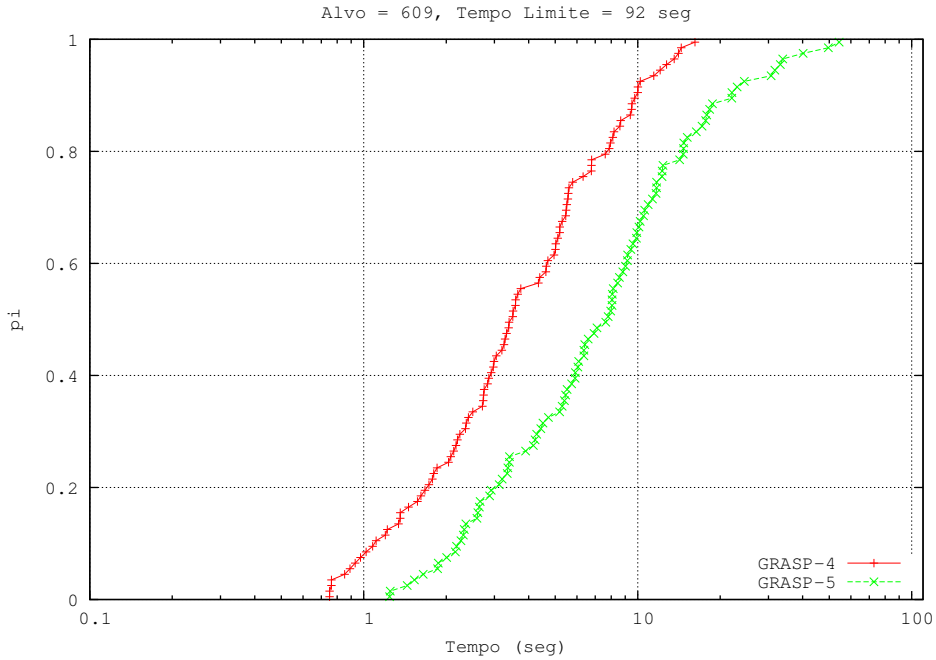


Figura 6.5: Comparação entre os algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1 (não conseguiu atingir a solução alvo) para a instância 7, solução alvo = 609 e tempo limite = 92 segundos.

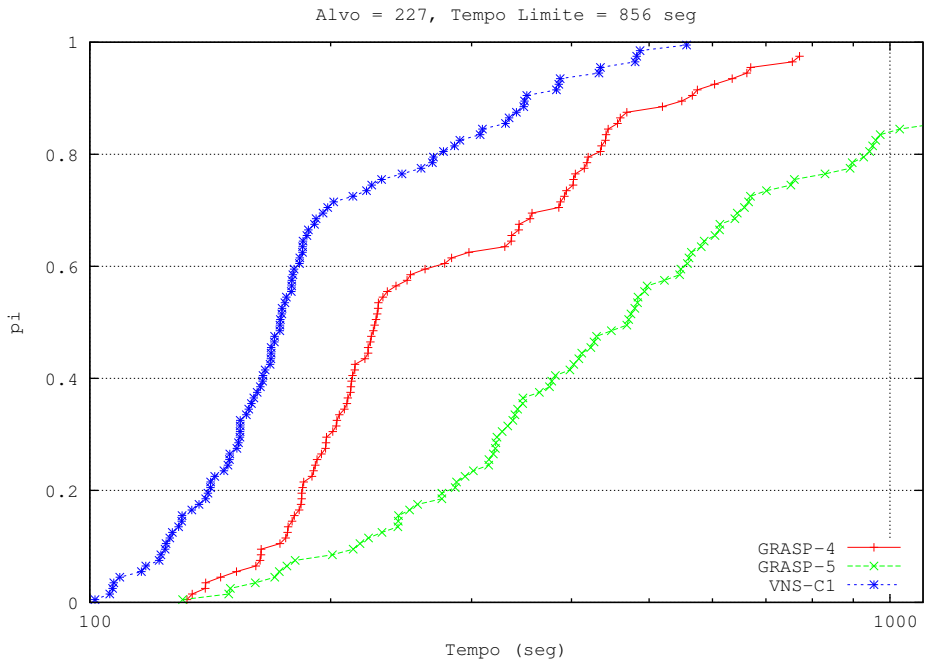


Figura 6.6: Comparação entre os algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1 para a instância 12, solução alvo = 227 e tempo limite = 856 segundos.

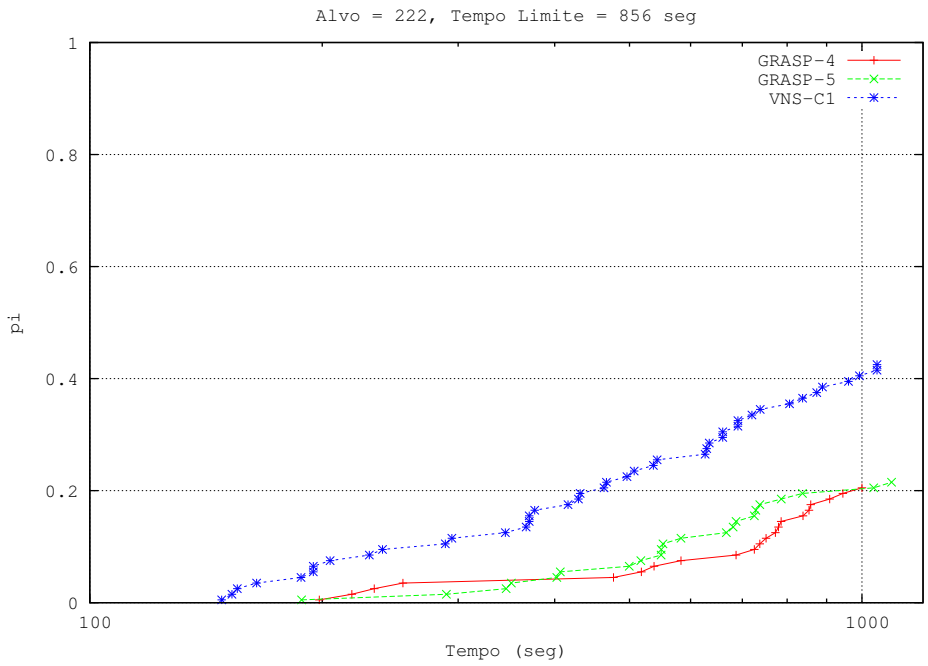


Figura 6.7: Comparação entre os algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1 para a instância 12, solução alvo = 222 e tempo limite = 856 segundos.

Pelos resultados das *Figuras 6.4* e *6.5* onde são comparados o desempenho do $GRASP\_4$, $GRASP\_5$ e VNS_1 nota-se que nestes testes o $GRASP\_4$ apresentou uma convergên-

cia empírica mais rápida para os valores alvos pré-estabelecidos. Nestes gráficos a curva relativa ao VNS_1 não está presente, já que para os dois casos este não conseguiu atingir o valor alvo em nenhuma execução.

O melhor desempenho do $GRASP\_4$ pode ser observado, por exemplo, na *Figura 6.5* onde este atinge uma convergência empírica de 100% em cerca de 16 segundos enquanto o $GRASP\_5$ exige cerca de 54 segundos para convergir esta taxa.

As *Figuras 6.6* e *6.7* ilustram o comportamento dos algoritmos $GRASP\_4$, $GRASP\_5$ e VNS_1 para uma instância maior (200 vértices), usando dois valores alvos ("fáciel"e "difícil").

Usando o alvo fácil(*Figura 6.6*) observamos que o VNS_1 se mostra a versão mais rápida, seguido pelo $GRASP\_4$ e do $GRASP\_5$. Neste exemplo note que o VNS_1 e $GRASP\_4$ conseguem em todas as execuções (curva atinge o nível 1 no eixo vertical) o que não ocorre com a versão $GRASP\_5$ que apresenta uma taxa de convergência de cerca de 80%.

A *Figura 6.7* ilustra a análise da mesma instância da *Figura 6.6*, mas usando um alvo mais difícil. Como conseqüência percebe-se a dificuldade dos três algoritmos em atingir a solução alvo. Neste sentido, o melhor desempenho foi novamente obtido pelo VNS_1 com uma taxa de convergência empírica de cerca de 40% enquanto os demais atingem somente taxa de 20%.

Capítulo 7

Conclusão

Este trabalho apresenta propostas para resolver de forma exata e aproximada uma variante do *Problema do Caixeiro Viajante - PCV* aqui denominada *Problema do Caixeiro Branco e Preto - PCVBP*. Embora esta variante seja aplicável em problemas de roteamento e escalonamento tais como escalonamento de aeronaves e tripulações aéreas, configurações de redes de telecomunicações [27] entre outros; identificamos na literatura apenas um único artigo sobre este problema [1]. Como contribuições são propostas: formulações matemáticas para o $PCVBP$ (caso assimétrico e simétrico) descrevendo-as como um problema de programação linear inteira, heurísticas do tipo $GRASP$, VND e VNS. Os testes computacionais efetuados mostram o potencial destes algoritmos, que nas instâncias pequenas, conseguem sempre atingir a solução ótima para os poucos casos onde esta é conhecida. Nas instâncias de maior porte, é verificado que versões híbridas conjugando $GRASP$ com busca local VND se mostraram mais promissoras.

Como propostas para trabalhos futuros sugerimos a utilização da metaheurística $GRASP$ com o reconexão de caminhos e o $GRASP$ reativo, bem como o uso de outras metaheurísticas como Busca Tabu e Algoritmos Genéticos.

Referências

[1] BOURGEOIS, M.; LAPORTE, G.; SEMET, F. Heuristic for the black and white traveling salesman problem. *Computers e Operations Research*, v. 30, p. 75–85, 2003.

[2] GENDREAU, M.; HERTZ, A.; LAPORT, G. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, v. 40, p. 1086–1094, 1992.

[3] GAREY, M. R. et al. *Some NP- Complete Problems*. [S.l.]: Eigth Annual Symp, on Theory of Computation, 1976. 10-22 p.

[4] E.GOLDEDBERG, D. *Genetic Algorithms in Search*. [S.l.]: Optimization and machine Learning, 1989. 10-22 p.

[5] KIRKPATRICK, S.; GELATT, C. D.; JR, M. P. V. Optimization by simulated annealing. *Science*, v. 220, p. 671–681, 1983.

[6] GLOVER, F. Tabu search - part i. *ORSA Journal on Computing*, v. 1, p. 190–206, 1989.

[7] FEO, T. A.; RESENDE, M. G. de C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

[8] GAMBARDELLA, L. M.; TAILLARD Éric; AGAZZI, G. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. *TECHNICAL REPORT IDSIA*, p. 06–99, 1999.

[9] HANSEN, P.; MLADENOVIC, N. Variable neighborhood search. *Computers and Operations Research*, v. 24, p. 1097–1100, 1997.

[10] LIN, S. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, v. 44, p. 2245–2269, 1965.

[11] LAWLER, E. L. et al. *The Traveling Salesman Problem - A Guided Tour of Combinatorial Otmization*. [S.l.]: John Wiley and Sons, 1995.

[12] GLOVER, F.; LAGUNA, M. Tabu search. Kluwer Academic, 1998.

[13] GENDREAU, M. Recent advancer in tabu search. Kluwer Academic, p. 369–379, 2002.

[14] RESENDE, M. G. de C.; RIBEIRO, C. da C. C. Greedy randomized adaptive search procedures. *Handbook of Metaheuristics*, Kluwer Academic, p. 219–249, 2003.

[15] HANSEN, P.; MLADENOVIC, N. Development od variable neighborhood search. *Essay and Surveys in Metaheuristics*, Kluwer Academic, p. 415–441, 2002.

[16] FEO, T. A.; RESENDE, M. G. de C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research Letters*, v. 8, p. 67–71, 1989.

[17] FEO, T. A.; RIBEIRO, C. da C. C. *Greedy randomized adaptive search procedures*. [S.l.]: Handbook of Metaheuristics, 2003. 219-249 p.

[18] PITSOULIS, S. L.; RESENDE, M. G. de C. *Greedy randomized adaptative search procedures*. [S.l.]: HANDBOOK OF APPLIED OPTIMIZATION, 2003. 168-183 p.

[19] PRAIS, M.; RIBEIRO, C. C. Variação de parâmetros em procedimentos grasp. *Investigación Operativa*, v. 9, p. 1–20, 2000.

[20] GLOVER, F. Tabu search and adaptive memory programing - advances, applications and challenges. *INTERFACES IN COMPUTER SCIENCE AND OPERATIONS RESEARCH*, Kluwer, p. 1–75, 1990.

[21] ROSSETI, I. C. M. *Estratégias seqüenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, 2003.

[22] MLADENOVIC, N. A variable neighborhood algorithm - a new metaheuristic for combinatorial optimization. *Presented of Days, Montreal*, 1995.

[23] HANSEN, P.; MLADENOVIC, N. Variable neighborhood search: Principles and aplications. *Operations Research*, v. 130, p. 449–467, 2001.

[24] HANSEN, P.; MLADENOVIC, N. *Variable neighbourhood search*. [S.l.]: Glover and Kochenagen, Kluwer, 2003. 145-184 p.

[25] RYAN, D. M.; HJORRING, C.; GLOVER, F. Extension of the petal method for the vehicle routing. *Journal of Operations Research Society*, v. 44, p. 289–296, 1993.

[26] AIEX, R. M.; RESENDE, M. G. de C.; RIBEIRO, C. da C. C. Probability distribution of solution time in grasp: An experimental investigation. *J. of Heurístics*, v. 8, p. 343–373, 2002.

[27] BASTOS, L.; MACAMBIRA, E. M.; OCHI, L. S. A relative neighbourhood grasp for the sonet ring assignment problem. Proc. of the INOC 2005, Book 3, INFORMS, v. 3, p. 833–838, 2005.