Universidade Federal Fluminense

MATHEUS DE SOUZA ALVES SILVA

Problema de Recobrimento de Rotas com Coleta de Prêmios

NITERÓI-RJ

2009

Universidade Federal Fluminense

MATHEUS DE SOUZA ALVES SILVA

Problema de Recobrimento de Rotas com Coleta de Prêmios

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 e Inteligência Artificial.

Orientador:
Prof. Luiz Satoru Ochi, D.Sc.

Co-Orientador:
Prof. Marcone Jamilson Freitas Souza, D.Sc.

NITERÓI-RJ

2009

Problema de Recobrimento de Rotas com Coleta de Prêmios

Matheus de Souza Alves Silva

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 e Inteligência Artificial.

Aprovada por:

Prof. Luiz Satoru Ochi, D.Sc. / IC-UFF
(Presidente) / (Orientador)

Prof. Marcone Jamilson Freitas Souza, D.Sc. / DECOM-UFOP
(Co-Orientador)

Profa. Marcia Helena Costa Fampa, D.Sc. / COOPE-UFRJ

Prof. Marcus Vinícius Soledade Poggi de Aragão, Ph.D. /
INF-PUC/RJ

Profa. Simone de Lima Martins, D.Sc. / IC-UFF

Niterói, 10 de junho de 2009.

À minha família e à Lorena, por todo o amor, carinho e apoio.

Agradecimentos

À Deus por sempre estar presente me iluminando e dando forças para alcançar meus objetivos.

Aos meus pais Geraldo e Regina, pelo amor, carinho, educação, apoio incondicional, incentivo e presença em todos os anos da minha vida.

Aos meus irmãos Eduardo e Ana Paula e à toda a família pelo apoio e amizade durante todos esses anos.

À Lorena, pelo amor, carinho e apoio que, mesmo distante, se faz presente em todos os dias.

À família Morimoto e Miyazaki pelos bons momentos de descontração.

Ao Instituto de Computação da Universidade Federal Fluminense, por ter me dado a oportunidade de desenvolver os estudos.

Aos professores e funcionários da UFF que foram muito prestativos durante todo o mestrado. Agradeço em especial, o professor Luiz Satoru Ochi pela orientação e amizade construída durante o mestrado.

Ao grande amigo e professor Marcone Jamilson Freitas Souza, da UFOP. Um pessoa ímpar que não mediu esforços para me ajudar e que mais uma vez esteve presente de forma decisiva em mais uma conquista na minha vida.

Aos amigos da Gapso pela amizade e grandes idéias que ajudaram para que esse trabalho se concretizasse.

Aos grandes amigos que me acolheram em Niterói, Rodrigo, Thiago, André e em especial meu irmão de república Daniel, pela amizade, companheirismo e bons momentos de confraternização.

Ao companheiros de república em Niterói, Marcio e Silas, de república no Rio de Janeiro, Thiago, Hadriel e Tomaz, por toda ajuda e amizade estabelecida durante esses

anos.

A todos os demais, que de alguma forma contribuíram para a realização deste trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro concedido.

Muito Obrigado!

Resumo

Este trabalho aborda uma generalização do Problema do Caixeiro Viajante (PCV), o chamado Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP). Este problema pode ser definido em um grafo não direcionado $G = (N, E)$, onde N representa o conjunto de vértices e E o conjunto de arestas. O conjunto N é formado por dois conjuntos disjuntos $N = (V \cup W)$, em que V é formado pela união dos vértices obrigatórios pertencentes a T com os vértices opcionais pertencentes a $V \setminus T$. A cada vértice i de T e $V \setminus T$ está associado um prêmio não-negativo p_i. O conjunto W é formado pelos vértices que precisam ser cobertos por algum vértice pertencente a V, isto é, deve existir pelo menos um vértice no conjunto V que faça parte da rota e esteja a uma distância igual ou inferior a D de um vértice de W. O PRRCP consiste em determinar uma rota de comprimento mínimo sobre um subconjunto de V, que contenha todos os vértices obrigatórios pertencentes a T e que todos os vértices de W sejam cobertos. Além disso, é necessário que uma quantidade mínima de prêmios, dada por *PRIZE*, seja coletada. Sendo uma generalização do PCV, o PRRCP é um problema NP-Difícil. Para sua resolução, este trabalho propõe novas regras de redução para o PRRCP, uma nova formulação matemática, um Algoritmo Evolutivo e cinco versões de um algoritmo heurístico baseado em *Iterated Local Search*. Para testar essas abordagens foi usado um conjunto de problemas-teste adaptados do PCV.

Palavras-chave: Problema de Recobrimento de Rotas com Coleta de Prêmios, Problema de Recobrimento de Rotas, Problema do Caixeiro Viajante, *Iterated Local Search*, Algoritmos Evolutivos, GENIUS.

Abstract

This work studied the generalization on the Travelling Salesman Problem which is known as The Prize Collecting Covering Tour Problem (PCCTP). This problem can be defined in a non-directed graph $G = (N, E)$, where N represents the set of nodes and E the set of arcs. The set N is composed by two disjunctive sets $N = (V \cup W)$, where V is formed by the union of the obligatory nodes in T with the optional nodes in $V \setminus T$. Each node i of V and $V \setminus T$ has a non-negative associated premium p_i. The set W is composed by the nodes that need to be covered by a node from V, it means that should exist a node V in the route and located in a distance equal or inferior to D of W. The PCCTP consists of determine a cycle of minimum length over a subset of V, which contain all obligatory nodes of T and all nodes of W have to be covered. Furthermore, it requires a minimal amount of awards, given by PRIZE to be collected. As a generalization of PCV, the PRRCP is an NP-hard problem. This work proposes new reduction rules for the PRRCP, a new mathematical formulation, an Evolutionary Algorithm and five versions of an heuristic algorithm based on Iterated Local Search. To test these approaches, a number of instances adapted from PCV has been used.

Keywords: Prize Collecting Covering Tour Problem, Covering Tour Problem, Traveling Salesman Problem, Iterated Local Search, Evolutionary Algorithms, GENIUS.

Siglas e Abreviações

AE	:	Algoritmo Evolutivo
CF	:	Colônia de Formigas
CTP	:	*Covering Tour Problem*
GCSP	:	*Geometric Covering Salesman Problem*
GRASP	:	*Greedy Randomized Adaptive Search Procedure*
ILS	:	*Iterated Local Search*
ILS-MRD	:	*ILS* tendo o *MRD* como método de busca local
MRD	:	Método Randômico de Descida
MTP	:	*Median Tour Problem*
MCTP	:	*Maximal Covering Tour Problem*
m-CTP	:	*Multi-Vehicle Covering Tour Problem*
PCCTP	:	*Prize Collecting Covering Tour Problem*
PCTSP	:	*Prize Collecting Traveling Salesman Problem*
PCV	:	Problema do Caixeiro Viajante
PPLI	:	Problema de Programação Linear Inteira
PRR	:	Problema de Recobrimento de Rotas
PRRCP	:	Problema de Recobrimento de Rotas com Coleta de Prêmios
PRRG	:	Problema de Recobrimento de Rotas Generalizado
RC	:	Reconexão por Caminhos
SCP	:	*Set Covering Problem*
SCPP	:	*Shortest Covering Path Problem*
STSP	:	*Selective Traveling Salesman Problem*
TSP	:	*Traveling Salesman Problem*
VND	:	*Variable Neighborhood Descent*
VNRD	:	*Variable Neighborhood Random Descent*
VNS	:	*Variable Neighborhood Search*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

1.1 O Problema de Recobrimento de Rotas com Coleta de Prêmios - PRRCP

Esta dissertação aborda uma generalização do Problema do Caixeiro Viajante (PCV), denominada Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP). O PRRCP pode ser definido em um grafo não direcionado $G = (N, E)$, com N representando o conjunto de vértices e $E = \{(i, j) \,|i, j \in N\}$, o conjunto de arestas. O conjunto N é formado por dois conjuntos disjuntos $N = (V \cup W)$, em que V é formado pela união dos vértices obrigatórios pertencentes a T, isto é, que devem fazer parte da rota, com os vértices que não necessariamente precisam fazer parte da rota, chamados de vértices opcionais pertencentes a $V \setminus T$. A cada vértice i de T e $V \setminus T$ está associado um prêmio não-negativo p_i. O conjunto W é formado pelos vértices que precisam ser cobertos por algum vértice pertencente a V, isto é, deve existir pelo menos um vértice no conjunto V que faça parte da rota e esteja a uma distância igual ou inferior a D de um vértice de W. Um prêmio mínimo pré-definido, dado por *PRIZE*, deve ser coletado. O PRRCP consiste em determinar uma rota de comprimento mínimo sobre um subconjunto de V, que contenha todos os vértices obrigatórios em T, que todos os vértices de W sejam cobertos e que a quantidade mínima de prêmios, *PRIZE*, seja coletada.

O PRRCP pode ser entendido como uma variação do Problema de Recobrimento de Rota (PRR), cuja definição é idêntica à do PRRCP, diferenciando-se apenas em relação à coleta de prêmios, que não é realizada no PRR. Outra variante do PRR fortemente relacionada ao PRRCP é o Problema de Recobrimento de Rota Generalizado (PRRG).

Esse problema consiste em determinar uma rota de comprimento mínimo em um subconjunto de vértices $V \cup W$, permitindo que os vértices de W façam parte da rota solução, ao contrário do PRR, cobrindo a si próprio e a outros vértices deste conjunto, quando for o caso.

O PRRCP tem uma aplicação possível na realização dos "Serviços de Atendimento Médico Móvel" (SAMM), que funciona da seguinte forma: um veículo sai de um ponto origem, visita um conjunto de pontos de atendimento (vértices de T e eventualmente alguns vértices do conjunto $V \setminus T$), de forma que a população de uma determinada região (conjunto W) seja contemplada com tais serviços sem que nenhuma pessoa tenha que se locomover mais que uma distância máxima D para chegar a um ponto de parada do veículo. Ao final, o veículo retorna ao local de origem. Os pontos de parada do veículo são definidos a partir de valores como: i) pontos de parada pré-estabelecidos (vértices de T), ii) pontos onde a quantidade de pessoas residentes na proximidade seja alta (prêmio p_i do vértice $i \in V$), e/ou que não estejam cobertas por pontos de T (vértices de $V \setminus T$).

De acordo com a literatura afim, existem poucos trabalhos relacionados ao PRRCP e suas variações. Para o PRRCP, tem-se a dissertação de mestrado de [Lyra, 2004], na qual é proposta uma formulação matemática e regras de redução para o mesmo. Em relação ao PRR, cita-se a tese de doutorado de [Current, 1981], que foi o trabalho pioneiro neste tema. Em [Gendreau *et al.*, 1995] são propostas uma formulação matemática e uma heurística para o PRR. No trabalho de [Maniezzo *et al.*, 1999] é proposta uma solução para o PRR através da aplicação de três algoritmos baseados na metaheurística Scatter Search [Glover, 1995], regras de redução para o PRR e uma formulação matemática. Para o PRRG, é de conhecimento a dissertação de mestrado de [Motta, 2001], em que é proposta uma formulação matemática, um conjunto de regras de redução e métodos heurísticos para sua resolução.

Por se tratar de uma generalização do PCV, este problema se enquadra na classe de problemas NP-difíceis. Assim, é desafiador o desenvolvimento de algoritmos eficientes para resolvê-lo.

1.2 Objetivos do trabalho

Este trabalho tem como objetivo geral o desenvolvimento de algoritmos de otimização eficientes para resolver o Problema de Recobrimento de Rotas com Coleta de Prêmios.

1.2.1 Objetivos específicos

Os seguintes os objetivos específicos deste trabalho são:

1. Efetuar uma revisão de literatura sobre o Problema de Recobrimento de Rotas com Coleta de Prêmios e problemas correlatos;
2. Efetuar uma breve revisão de técnicas heurísticas de otimização;
3. Desenvolver uma nova formulação de programação matemática para o PRRCP;
4. Desenvolver algoritmos heurísticos para resolver o PRRCP, utilizando os conceitos da metaheurística *Iterated Local Search* e outra baseada em heurística populacional;
5. Gerar um conjunto de problemas-teste para o PRRCP;
6. Comparar e avaliar empiricamente as abordagens propostas.

1.3 Estrutura do trabalho

O presente trabalho está dividido em 9 capítulos, incluido esta introdução.

No Capítulo 2 é realizado um levantamento da literatura sobre o PRR, PRRG e o PRRCP, bem como as heurísticas e metaheurísticas utilizadas na resolução do problema. No Capítulo 3 descreve-se detalhadamente o Problema de Recobrimento de Rotas com Coleta de Prêmios e apresentam-se alguns problemas similares ao PRRCP. Apresenta-se, no Capítulo 4, uma formulação matemática existente na literatura para o PRRCP e propõe-se uma nova formulação para o mesmo. No capítulo 5 são mostradas as regras de redução para o PRR e PRRCP existentes na literatura e apresentam-se também novas regras de redução propostas neste trabalho. O Capítulo 6 mostra como as heurísticas apresentadas no Capítulo 3 foram adaptadas para serem aplicadas ao PRRCP. Por se tratar de um problema relativamente novo na literatura, o Capítulo 7 apresenta um gerador de problemas-teste para o PRRCP, a partir de uma adaptação dos problemas disponíveis na literatura para o PCV. Por fim, os Capítulos 7 e 8 apresentam os resultados computacionais e as conclusões respectivamente. Como o número de problemas-teste utilizados é muito grande e por não existir nenhuma informação sobre eles na literatura, foi disponibilizado no endereço http://www.ic.uff.br/∼satoru/conteudo/artigos/Resultados%20Problemas-teste%20PRRCP.pdf informações mais detalhadas dos problemas-teste, das regras de redução e dos resultados obtidos.

Capítulo 2

Revisão Bibliográfica

2.1 Introdução

O Problema de Recobrimento de Rotas (PRR), conhecido na literatura como *The Covering Tour Problem*, é uma generalização do Problema do Caixeiro Viajante (PCV). Dessa forma, a obtenção da solução ótima por meio de métodos exatos de programação matemática está limitada, via de regra, a problemas de pequenas dimensões. Para dimensões mais elevadas, a abordagem mais comum é através de técnicas heurísticas. Essas técnicas apresentam como grande vantagem a flexibilidade na manipulação do problema, permitindo incluir, com maior facilidade, qualquer espécie de restrição do problema. A principal desvantagem dessas técnicas é não se saber a qualidade da solução encontrada. Entretanto, com o aparecimento das chamadas metaheurísticas, ou heurísticas inteligentemente flexíveis, tais como Busca Tabu [Glover, 1986], Simulated Annealing [Kirkpatrick *et al.*, 1983], GRASP [Feo e Resende, 1995], Algoritmos Genéticos [Goldberg, 1989] e Método de Busca em Vizinhança Variável [Mladenovic e Hansen, 1997], que possuem mecanismos para tentar escapar de ótimos locais ainda distantes de um ótimo global, muito progresso se tem conseguido com o objetivo de melhorar a qualidade das soluções finais geradas.

Apesar de possuir várias aplicações possíveis, o PRR não tem sido muito estudado pelos pesquisadores, desde que foi introduzido, em 1981, por Current [Current, 1981].

Em [Current e Schilling, 1989] e [Current e Rolland, 1994] é desenvolvida uma heurística que gera um conjunto de soluções para PRR em duas etapas. Na primeira, minimiza-se o comprimento da rota. Na segunda, maximiza-se o número de vértices cobertos pela rota gerada na primeira etapa.

[Gendreau *et al.*, 1995] apresentam e analisam uma formulação matemática para o PRR. Para definir o modelo, considere as seguintes notações:

- V: conjunto dos vértices que podem fazer parte da rota, $V = T \cup V \setminus T$;
- W: conjunto dos vértices que devem ser cobertos por algum vértice de V que faça parte da solução;
- T: conjunto de vértices obrigatórios;
- $V \setminus T$: conjunto de vértices opcionais;
- S: subconjunto de V, tal que $2 \leq |S| \leq n-2, V \setminus S \neq \emptyset$;
- $S_l = i \in V : c_{ij} \leq D, l \in W$.

Além das notações apresentadas, sejam as seguintes variáveis de decisão do problema:

- y_k: variável binária que assume valor 1 se o vértice $k \in V$ faz parte da rota e 0, caso contrário;
- x_{ij}: variável binária que assume valor 1 se a aresta (i, j), com $i, j \in V$ e $i < j$, faz parte da rota e 0, caso contrário.

De acordo com a definição do PRR, os vértices do subconjunto T são obrigados a fazer parte da rota. Assim, se $k \in T$, tem-se necessariamente $y_k = 1$.

De posse dessas considerações, o PRR pode ser formulado pelas seguintes equações:

$$\text{(P1)} \qquad \text{Minimizar} \sum_{i<j} c_{ij} x_{ij} \tag{2.1}$$

$$\text{s.a:} \qquad \sum_{k \in S_l} y_k \geq 1 \qquad \forall l \in W \tag{2.2}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2y_k \qquad \forall k \in V \tag{2.3}$$

$$\sum_{(i,j)\,|\,i \in S, j \in V \setminus S} x_{ij} \geq 2y_t \qquad \forall t \in S \tag{2.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall i, j \in V, i < j \tag{2.5}$$

$$y_k = 1 \qquad \forall k \in T \tag{2.6}$$

$$y_k \in \{0,1\} \qquad \forall k \in V \setminus T \tag{2.7}$$

Nesta formulação, a função (2.1) procura minimizar o custo total da rota. As restrições (2.2) garantem que todos os vértices de W serão cobertos, enquanto as restrições (2.3) determinam o grau dos vértices de V, isto é, se um vértice k estiver na rota, então é necessário que ela tenha grau 2. As restrições (2.4) garantem a conectividade da rota, forçando a presença de pelo menos duas arestas entre quaisquer subconjuntos S e $V \setminus S$ de forma que S esteja contido em V e contenha um vértice t pertencente à rota. Além disso $V \setminus S \neq \emptyset$. Para cada circuito hamiltoniano possível é necessário uma restrição do tipo (2.4), o que justifica o número de $O(2^n)$ restrições. As restrições (2.5) e (2.7) garantem a integralidade das variáveis x_{ij} e y_k respectivamente e, por fim, as restrições (2.6) garantem que todos os vértices obrigatórios estarão na rota.

Esta formulação exige um número exponencial de restrições para evitar a formação de subrotas desconexas da origem, no caso, $O(2^n)$, tanto no PRR quanto no PRRCP.

Além dessa formulação, também foram propostos nesse trabalho um algoritmo exato baseado na técnica *Branch and Cut* e um procedimento heurístico que combina o algoritmo denominado PRIMAL1 *Set Covering*, proposto por [Balas e Ho, 1980], com a heurística GENIUS [Gendreau *et al.*, 1992]. A melhor solução retornada pela heurística é utilizada como limite superior inicial para o algoritmo exato.

[Maniezzo *et al.*, 1999] propõem três algoritmos baseados na metaheurística *Scatter Search* [Glover, 1995] para resolver o PRR. A principal característica desta metaheurística é a utilização de um conjunto, chamado *Conjunto Referência*, para armazenar as R melhores soluções encontradas. Operadores de recombinação são aplicados entre as soluções do *Conjunto Referência* gerando novas soluções que posteriormente são refinadas por um processo de busca local.

Além dos algoritmos propostos, [Maniezzo *et al.*, 1999] propõem um modelo de programação matemática para o PRR, adaptado de uma formulação proposta originalmente por [Finke *et al.*, 1984] para o PCV. Esse modelo descreve o PRR como um problema de programação linear inteira, associando duas variáveis de fluxo x_{ij} e x_{ji} a cada aresta (i, j) do grafo. Se a rota vai do vértice i ao j, então x_{ij} representa o número de vértices que podem ser visitados e x_{ji} representa o número de vértices já visitados. Dessa forma, a variável x_{ij} define dois circuitos para qualquer rota que representar uma solução viável, enquanto o segundo circuito é definido pelas variáveis de fluxo representando o número de vértices já visitados. Além das variáveis de fluxo, tem-se as variáveis binárias ξ_{ij} que indica se a aresta $(i, j) \in E$ está presente na solução e y_i que indica se o vértice $i \in V$ foi visitado ou não.

As equações (2.8) a (2.16) descrevem a modelo dos autores para o PRR.

$$\text{(P2)} \qquad \text{Minimizar} \sum_{(i<j)|i,j\in V} c_{ij}\xi_{ij}, \tag{2.8}$$

$$\text{s.a:} \qquad \sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = -2y_i \qquad \forall i \in V \tag{2.9}$$

$$\sum_{j\in V}(x_{ij} + x_{ji}) = 2y_i(|V| - 1) \qquad \forall i \in V \tag{2.10}$$

$$\sum_{i\in S_l} y_i \geq 1 \qquad \forall l \in W \tag{2.11}$$

$$x_{ij} + x_{ji} = (n-1)\xi_{ij} \qquad \forall (i,j) \in E \tag{2.12}$$

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

$$y_i = 1 \qquad \forall j \in T \tag{2.14}$$

$$y_j \in \{0,1\} \qquad \forall j \in (V \setminus T) \tag{2.15}$$

$$\xi_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{2.16}$$

Nesta formulação, as restrições (2.9) e (2.13) definem um fluxo viável para as variáveis x_{ij}. As restrições (2.10) e (2.12) garantem que todo vértice i que esteja na rota possua grau 2. As restrições (2.11), nas quais, $S_l = \{i \in V : d_{il} \leq D,\ l \in W\}$, asseguram a cobertura de todos os vértices de W, garantindo que pelo menos um vértice $i \in V$, tal que $d_{il} \leq D$, esteja na rota. Finalmente, as restrições (2.14) e (2.16) referem-se às condições de integralidade das variáveis y_j e ξ_{ij}.

Em sua dissertação, [Kubik, 2007] propõe uma técnica baseada em Colônia de Formigas - CF [Dorigo, 1992] para resolver o PRR. A abordagem proposta divide o PRR em dois outros problemas, o *Traveling Salesman Problem* (TSP) e o *Set Covering Problem* (SCP). O objetivo do TSP é construir uma rota de custo mínimo sobre o conjunto de vértices obrigatórios, enquanto que no SCP determina-se um conjunto de colunas que cobrem um conjunto de linhas com um custo mínimo. Nesse problema, as linhas representam os vértices que precisam ser cobertos (W), enquanto as colunas se referem aos vértices que podem cobrir os vértices do conjunto W, isto é, os vértices dos conjuntos T e $V \setminus T$. No algoritmo proposto por [Kubik, 2007], resolve-se inicialmente o SCP utilizando-se o procedimento Colônia de Formigas clássico, que utiliza a heurística do vizinho mais próximo para definir o próximo vértice a entrar na solução. Ao final da resolução do SCP, utiliza-se os vértices da melhor solução encontrada como problema-teste para o TSP. Esse problema é resolvido utilizando-se também a técnica Colônia de Formigas, diferenciando-se do an-

terior no método de inserção dos vértices na solução. Nesse procedimento, esses vértices são inseridos utilizando-se a fase GENI do algoritmo GENIUS.

Proposto em [Motta, 2001], o Problema de Recobrimento de Rotas Generalizado (PRRG) consiste em encontrar uma rota de comprimento mínimo em um subconjunto de vértices de $V \cup W$, permitindo que, ao contrário do PRR, todos os vértices de W façam parte da rota solução, cobrindo a si próprio e a outros vértices desse conjunto, quando for o caso.

Para resolução do PRRG, são propostos em [Motta *et al.*, 2001a, Motta *et al.*, 2001b, Motta *et al.*, 2001c] uma formulação matemática utilizando variáveis de fluxo, um conjunto de regras de redução, bem como um algoritmo heurístico baseado nas técnicas GRASP [Feo e Resende, 1995] e VNS [Mladenovic e Hansen, 1997].

Na formulação de programação matemática proposta, o número de restrições é polinomial em relação ao tamanho do problema, ao contrário de algumas formulações da literatura, que exigem um número exponencial de restrições associadas ao PRR. Tomando-se como base a definição do PRR descrita anteriormente, a idéia básica da formulação desses autores é associar uma variável de fluxo a cada aresta $(i, j) \in E$. Para apresentá-la, sejam as seguintes variáveis:

- y_k: variável binária que indica se o vértice $k \in V$ faz parte da rota ou não. Se $k \in T$, então y_k é necessariamente igual a 1;
- x_{ij}: variável binária que indica se a aresta (i, j), com $i, j \in V \cup W$ e $i \neq j$, faz parte da rota ou não;
- z_{ij}: variável inteira não-negativa que representa a quantidade de fluxo que passa na aresta $(i, j) \in E$, com $i \neq j$.

Também são definidos os coeficientes binários δ_{lk}, que indicam se o vértice $l \in W$ pode ser coberto pelo vértice $k \in V$, isto é, se $d_{lk} \leq D$ e $S_l = k \in V \mid \delta_{lk} = 1$.

De posse destas considerações, a formulação proposta é descrita da seguinte forma:

$$\text{(P3)} \qquad \text{Minimizar} \sum_{i<j|i,j\in V} c_{ij}x_{ij} \tag{2.17}$$

$$\text{s.a:} \sum_{k\in S_l} y_k \geq 1 \qquad \forall l \in W \tag{2.18}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2y_k \qquad \forall k \in (V \cup W) \tag{2.19}$$

$$\sum_{j\in V\cup W} z_{kj} = \sum_{i\in V\cup W} x_{ik} + 2y_k \qquad \forall k \in (V \cup W) \setminus \{s\} \tag{2.20}$$

$$\sum_{j\in(V\cup W)} z_{sj} = 1 \tag{2.21}$$

$$\sum_{j\in(V\cup W)} z_{js} = \sum_{j\in(V\cup W)} y_j \tag{2.22}$$

$$x_{ij} \leq z_{ij} \qquad \forall i,j \in (V \cup W) \tag{2.23}$$

$$x_{ij} \geq (z_{ij})/(|V| + |W| + 1) \qquad \forall i,j \in (V \cup W) \tag{2.24}$$

$$y_k = 1 \qquad k \in T \tag{2.25}$$

$$y_k \in \{0,1\} \qquad k \in (V \setminus T) \tag{2.26}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in (V \cup W) \tag{2.27}$$

$$z_{ij} \in Z^+ \qquad \forall i,j \in (V \cup W) \tag{2.28}$$

Nesta formulação, as restrições (2.18) asseguram que cada vértice de W é coberto por pelo menos um vértice da rota. As restrições (2.19) são responsáveis pela conservação de fluxo, enquanto as restrições (2.20) a (2.22) asseguram a conectividade da rota. As restrições (2.23) e (2.24) garantem que a rota gerada a partir da variável de fluxo z_{ij} e a variável de decisão x_{ij} se coincidem. As restrições (2.25) asseguram que todos vértices de T farão parte da rota. Por fim, as restrições (2.26) a (2.28) representam as condições de integralidade do problema.

Além da formulação de programação matemática, foram propostas duas regras de redução para o PRRG. Essas regras têm como objetivo a eliminação de informações que são consideradas irrelevantes à resolução do problema. Com essa possível redução do espaço de busca, reduz-se, também, o tempo consumido pelos algoritmos propostos para resolvê-lo. Segundo os autores, os testes apresentados em [Gendreau *et al.*, 1995] e [Maniezzo *et al.*, 1999] para a simplificação do espaço de busca de uma instância do PRR não funcionam para o PRRG.

Para definir essas regras, seja Cob uma matriz binária de dimensões $|W| \times |V \cup W|$,

cujos elementos $cob_{ij} = 1$, $i \in W$, $j \in (V \cup W)$ se e somente se $c_{ij} \leq D$. Os conjuntos W e V podem ser reduzidos utilizando as seguintes regras:

1. Transformar cada vértice $i \in W$ em um vértice $l \in V \setminus T$, se $cob_{ij} = 1$, com $j \in T$;
2. Remover todo vértice $i \in V \setminus T$, tal que $cob_{ji} = 0$, $\forall j \in W$.

A primeira regra de redução diz que se existe um vértice $j \in T$ que cubra um vértice $i \in W$, então esse vértice i poderá ser utilizado para cobrir outros vértices do conjunto W. Dessa forma, o vértice i passa a pertencer ao conjunto $V \setminus T$. Já a segunda regra remove todos os vértices opcionais que não cobrem nenhum vértice do conjunto W.

Para testar o quanto a utilização das regras de redução diminui o tempo de execução, os autores aplicaram o algoritmo heurístico baseado em GRASP e VNS.

As instâncias utilizadas para teste foram geradas aleatoriamente com diferentes percentagens de $|T|$ e $|W|$. De acordo com os resultados obtidos, verificou-se que as regras de redução diminuíram o tamanho do grafo associado entre 40% a 60%, na média. Observou-se, também, que as soluções finais obtidas pelo algoritmo heurístico com regras de redução, foram em média 60% melhores que aquelas encontradas pelo algoritmo sem tais regras.

Em relação ao PRRCP, por se tratar de um tema recente, a dissertação de [Lyra, 2004] foi o único trabalho encontrado na literatura. Nesse trabalho, são propostas novas regras de redução para o PRRCP, uma formulação matemática e uma abordagem heurística simples baseada na metaheurística GRASP. Sendo este o tema do presente trabalho, as propostas de [Lyra, 2004] serão melhor detalhadas nos capítulos subseqüentes.

2.2 Heurísticas

As heurísticas são técnicas que visam a obtenção de soluções de boa qualidade (próximas da otimalidade) em um tempo computacional razoável. Essas técnicas, no entanto, não garantem a obtenção da solução ótima para o problema nem são capazes de garantir o quão próximo a solução obtida está da ótima.

As heurísticas podem ser construtivas ou de refinamento. As construtivas tem por objetivo construir uma solução, usualmente, elemento a elemento. A escolha de cada elemento está, geralmente, relacionada a uma determinada função que o avalia de acordo com sua contribuição para a solução. Tal função varia de acordo com o problema abordado. Nas heurísticas clássicas, os elementos candidatos a fazer parte da solução são,

geralmente, ordenados seguindo uma função gulosa, que estima o benefício da inserção de cada elemento e somente o elemento que trouxer o maior ganho para a solução é inserido a cada passo.

As heurísticas de refinamento, ou também chamados de heurísticas de busca local, são técnicas baseadas na noção de vizinhança. Para se definir o que é uma vizinhança, seja S o espaço de pesquisa de um problema de otimização e f a função objetivo a minimizar. O conjunto $N(s) \subseteq S$, o qual depende da estrutura do problema tratado, reúne um número determinado de soluções s', denominado vizinhança de s. Cada solução $s' \in N(s)$ é chamada de *vizinho* de s e é obtido de s a partir de uma operação chamada de *movimento*. Uma busca local em $N(s)$ tem como objetivo analisar todos os elementos de $N(s)$ para que, ao final da busca, se encontre um ótimo local em $N(s)$.

As seções 2.2.1 e 2.2.2 descrevem, respectivamente, as heurísticas de refinamento utilizadas neste trabalho, a saber, o Método Randômico de Descida e o Método de Descida Randômica em Vizinhança Variável.

2.2.1 Método Randômico de Descida

O Método Randômico de Descida, ou MRD, é uma heurística de refinamento que, a partir de uma solução inicial, analisa um vizinho qualquer e o aceita somente se ele for estritamente melhor que a solução corrente. Caso esse vizinho não seja melhor, a solução corrente permanece inalterada e outro vizinho é gerado. O procedimento é finalizado quando se atinge um número máximo de iterações sem que haja melhora no valor da melhor solução obtida. Não há garantia que essa solução retornada ao final do procedimento seja um ótimo local, uma vez que nem toda a vizinhança é explorada.

O pseudocódigo do MRD é apresentado no Algoritmo 1.

Algoritmo 1 Método Randômico de Descida

```
1: Defina uma solução inicial s_0 e faça s ← s_0;
2: iter ← 0;
3: enquanto iter < iterMRD faça
4:    iter ← iter + 1;
5:    Gere aleatoriamente um vizinho s' ∈ N(s);
6:    se f(s') < f(s) então
7:       s ← s';
8:       iter ← 0;
9:    fim se
10: fim enquanto
11: Retorne s;
```

2.2.2 Método de Descida Randômica em Vizinhança Variável

O Método de Descida Randômica em Vizinhança Variável (*Variable Neighboohoord Random Descent*, VNRD), é uma variante do Método de Descida em Vizinhança Variável (*Variable Neighboohoord Descent*, VND). O VND [Mladenovic e Hansen, 1997] é um método de refinamento que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança. Diferentemente do VND, no VNRD a exploração de cada vizinhança não é feita por completa, apenas parte desta é explorada. O pseudocódigo do VNRD é apresentado no Algoritmo 2.

Como se observa no Algoritmo 2, são consideradas r diferentes estruturas de vizinhança $\{N^{(1)}, N^{(2)}, \cdots, N^{(r)}\}$. A busca é iniciada a partir de uma solução s usando-se a primeira estrutura de vizinhança $N^{(1)}$. A seguir, escolhe-se um vizinho s' qualquer na vizinhança corrente de s e verifica-se se este é melhor que s. Em caso positivo, move-se para esse vizinho e repete-se o procedimento; caso contrário, volta-se para a solução corrente s e escolhe-se outro vizinho aleatório. Esta descida randômica na vizinhança corrente é interrompida após $iterMAX$ iterações sem melhora na solução corrente. Se a exploração na vizinhança corrente resultou em uma melhora, retorna-se à primeira vizinhança; caso contrário, passa-se para a próxima vizinhança. O algoritmo VNRD termina após explorar $iterMAX$ vizinhos de cada uma das r vizinhanças sem sucesso.

Algoritmo 2 Método de Descida Randômica em Vizinhança Variável

```
1: Seja s uma solução inicial;
2: Seja r o número de estruturas diferentes de vizinhança;
3: s* ← s; k ← 1;
4: enquanto k ≤ r faça
5:    iter ← 0;
6:    enquanto iter < iterMAX faça
7:       iter ← iter + 1;
8:       Gere aleatoriamente um vizinho s' ∈ N^(k)(s);
9:       se f(s') < f(s) então
10:         s ← s'; iter ← 0;
11:      fim se
12:   fim enquanto
13:   se f(s) < f(s*) então
14:      s* ← s; k ← 1;
15:   senão
16:      k ← k + 1;
17:   fim se
18: fim enquanto
19: Retorne s*;
```

2.3 Metaheurísticas

As metaheurísticas, ao contrário das heurísticas convencionais, são procedimentos genéricos de busca local que exploram o espaço de soluções, com capacidade de escapar das armadilhas dos ótimos locais ainda distantes de um ótimo global. Esses procedimentos são destinados a resolver aproximadamente um problema de otimização.

De acordo com o princípio utilizado para explorar o espaço de soluções, as metaheurísticas podem ser classificadas como sendo de busca local ou de busca populacional.

Nas metaheurísticas baseadas em busca local, o espaço de soluções é explorado por meio de movimentos aplicados em cada iteração sobre uma solução corrente, gerando soluções vizinhas promissoras. Já os métodos baseados em busca populacional consistem em manter um conjunto de boas soluções e combiná-las de diferentes maneiras a fim de se produzir uma nova população de soluções ainda melhores. Neste contexto, estas técnicas são também conhecidas como Heurísticas Evolutivas.

Na seções 2.3.1 e 2.3.2 são apresentadas as metaheurísticas referenciadas neste trabalho, a saber, GRASP e *Iterated Local Search*, baseadas no mecanismo de busca local, e na Seção 2.3.3 são apresentados os Algoritmos Evolutivos, os quais mesclam os conceitos de busca local e busca populacional.

2.3.1 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure* - Procedimento de Busca Adaptativa Randômica e Gulosa) é um método iterativo, proposto por [Feo e Resende, 1995], que procura conjugar bons aspectos dos algoritmos puramente gulosos com aqueles dos procedimentos aleatórios de construção de soluções.

O GRASP é constituído de duas fases, uma de construção e outra de busca local. Na primeira, uma solução é iterativamente construída, elemento a elemento; na segunda, essa solução construída é refinada. Essas duas fases são executadas por um determinado número de iterações. O Algoritmo 3 ilustra o procedimento básico da metaheurística GRASP.

Algoritmo 3 GRASP básico

1: $s^* \leftarrow \emptyset \Rightarrow f(s^*) = +\infty$;
2: **para** $Iter = 1, 2, ..., GRASPmax$ **faça**
3: $s_0 \leftarrow Construcao(\alpha)$;
4: $s \leftarrow BuscaLocal(s_0)$;
5: **se** $f(s) < f(s^*)$ **então**
6: $s^* \leftarrow s$;
7: **fim se**
8: **fim para**
9: Retorne s^*;

A fase de construção é dita iterativa e adaptativa. Ela é iterativa porque uma solução é construída, a cada passo, elemento por elemento. É adaptativa porque os benefícios associados com a escolha de cada elemento são atualizados a cada iteração da fase de construção para refletir as mudanças causadas pela seleção do elemento anterior. A componente probabilística do procedimento está relacionada à seleção de cada elemento em uma lista restrita de candidatos (LRC), subconjunto da lista C de elementos ainda não inseridos na solução parcial. O tamanho da LRC é determinado por um parâmetro α que controla a aleatoriedade do procedimento. Quando $\alpha = 0$, o procedimento de construção torna-se totalmente "guloso", pois restringe a lista de candidatos a apenas um elemento. Quando $\alpha = 1$, a construção torna-se totalmente aleatória, já que todos os elementos remanescentes são candidatos a ser escolhidos como próximo elemento a entrar na solução. Uma função "gulosa" $g : C \rightarrow \mathbb{R}$, mensura o benefício de se selecionar cada elemento candidato do conjunto C. O Algoritmo 4 ilustra o procedimento da fase de construção da metaheurística GRASP para um problema de minimização.

Algoritmo 4 Fase de Construção GRASP

1: $s \leftarrow \emptyset$;
2: Inicializar a lista de candidatos C;
3: **enquanto** $C \neq \emptyset$ **faça**
4: $g_{min} = \min\{g(t) \mid t \in C\}$;
5: $g_{max} = \max\{g(t) \mid t \in C\}$;
6: LRC $= \{t \in C \mid g(t) \leq g_{min} + \alpha\,(g_{max} - g_{min})\}$;
7: Selecionar $t \in LRC$ aleatoriamente;
8: $s = s \cup \{t\}$;
9: **fim enquanto**
10: Retorne s;

A fase de busca local consiste no refinamento da solução construída, já que a solução gerada na primeira fase não representa necessariamente um ótimo local. Um algoritmo de busca local típico consiste na substituição da solução atual pela melhor solução de sua

vizinhança. Na maioria dos casos, a busca se encerra quando nenhuma solução de melhor qualidade é encontrada na vizinhança. Os principais aspectos responsáveis pelo sucesso de uma busca local consistem na escolha eficiente da estrutura de vizinhança, na eficiência da técnica de busca empregada nesta vizinhança e também da qualidade da solução inicial. O Algoritmo 5 apresenta o pseudocódigo da fase de busca local para um problema de minimização.

Algoritmo 5 Fase de Busca Local GRASP

1: $V = \{s' \in N(s) \mid f(s') < f(s)\}$;
2: **enquanto** $|V| > 0$ **faça**
3: Selecione o melhor vizinho $s' \in V$ segundo a função f;
4: $s \leftarrow s'$;
5: $V = \{s' \in N(s) \mid f(s') < f(s)\}$;
6: **fim enquanto**
7: Retorne s;

2.3.2 *Iterated Local Search*

O *Iterated Local Search* - ILS [Stützle e Hoos, 1999], é um método de busca local que procura focar a busca não no espaço completo de soluções, mas em um subespaço definido por soluções que são ótimas locais segundo um determinado mecanismo de otimização. De acordo com [Lourenço *et al.*, 2003], o sucesso do ILS é centrado no conjunto de amostragem de ótimos locais, juntamente com a escolha da busca local, das perturbações e do critério de aceitação de uma solução.

Esse método funciona da seguinte maneira: seja f a função custo do problema a ser minimizada, S o conjunto de todas as possíveis soluções e s uma solução candidata.

O objetivo no ILS é explorar S^*, um subconjunto de S contendo apenas ótimos locais, caminhando-se de um ótimo local s^* para outro próximo a ele. Para cumprir esse objetivo, dado um ótimo local s^*, aplica-se uma mudança ou perturbação de forma a gerar uma solução intermediária s' pertencente a S. Em seguida, aplica-se um procedimento de busca local em s', gerando-se um ótimo local $s^{*\prime}$ em S^*. Se $s^{*\prime}$ passar no teste de aceitação, ele se torna o próximo elemento da caminhada em S^*; senão, retorna-se à s^*. Esse procedimento pode encontrar regiões promissoras no espaço de soluções. Para isso, a intensidade das perturbações não pode ser nem tão baixa e nem tão alta. Se a intensidade for baixa, s' pode pertencer à região de atração de s^* e com isso poucas novas soluções de S^* serão exploradas. Por outro lado, se a intensidade for muito alta, s' será uma solução aleatória e o algoritmo se comportará como um procedimento de múltiplos reinícios aleatórios.

Geralmente a caminhada do ILS não é reversível, isto é, se a caminhada for de s_1 para s_2, não se consegue vir de s_2 para s_1.

Como perturbações determinísticas podem conduzir a formação de ciclos, pode-se tornar a perturbação aleatória ou adaptativa para tentar evitar a ciclagem.

O Algoritmo 6 mostra o pseudocódigo do algoritmo ILS básico.

Algoritmo 6 *Iterated Local Search* básico

1: $s_0 \leftarrow GeraSolucaoInicial()$;
2: $s^* \leftarrow BuscaLocal(s_0)$;
3: **enquanto** *critério de parada não satisfeito* **faça**
4: $s' \leftarrow Perturbacao(histórico, s^*)$;
5: $s^{*\prime} \leftarrow BuscaLocal(s')$;
6: $s^* \leftarrow CriterioAceitacao(histórico, s^*, s^{*\prime})$;
7: **fim enquanto**
8: Retorne s^*;

2.3.3 Algoritmos Evolutivos

O termo Algoritmo Evolutivo (AE) compreende uma família de resolvedores estocásticos de problemas com base em princípios que podem ser encontrados na evolução biológica. Dentro desse contexto, encontrar uma solução para um dado problema é encarado como uma luta pela sobrevivência: possíveis soluções competem umas com as outras pela sobrevivência e pelo direito de reproduzir. Essa competição é a força motriz do progresso que supostamente conduz a uma solução ótima [Eiben e Rudolph, 1999].

Apesar de as idéias da computação evolutiva existirem desde a década de 1960, essa área teve maior impulso a partir do livro de John Holland, intitulado *Adaptation in Natural and Artificial Sytems*, publicado em 1975 [Reeves, 2003].

As três linhas principais de estudo nessa área são os Algoritmos Genéticos (AGs), Estratégias de Evolução (EE) e Programação Evolutiva (PE). O termo algoritmo evolutivo foi proposto em 1990, concebida como uma superclasse contendo todas as variantes acima referidas e também todas as outras técnicas baseadas na percepção evolutiva sobre resolução de problemas [Eiben e Rudolph, 1999].

Um AE típico funciona da seguinta forma: a cada geração, constrói-se um conjunto de indivíduos, representando as soluções de um problema de otimização. Esses indivíduos, juntamente com a população vinda de gerações anteriores, são combinados (fase de cooperação) para formarem novos indivíduos. Esses novos indivíduos passam, então, por uma

fase de adaptação antes de se decidir quais serão incluídos na população que irá para a próxima geração. O algoritmo é finalizado após um certo número de gerações, dado por *MAXgerações*, retornando-se o melhor indivíduo encontrado durante o processo evolutivo [Hertz e Kobler, 2000].

O Algoritmo 7 apresenta o pseudocódigo geral de um Algoritmo Evolutivo.

Algoritmo 7 Algoritmo Evolutivo básico

1: $P \leftarrow GeraPopulaçãoInicial()$;
2: **para** $geração = 1, 2, \cdots, MAXgerações$ **faça**
3: $\quad Cooperação(P)$;
4: $\quad Adaptação(P)$;
5: $\quad P^* \leftarrow AtualizaMelhoresIndivíduos(P)$;
6: **fim para**
7: Retorne $s^* \in P^*$;

Durante o período de cooperação, as informações dos indivíduos da população são trocadas entre os reprodutores. Cada novo indivíduo é formado por uma associação de indivíduos da população, sendo que cada indivíduo poderá ou não transmitir informações. Os operadores de seleção e de cruzamento (*crossover*) dos Algoritmos Genéticos tradicionais podem ser considerados procedimentos de cooperação. A troca de informação entre os indivíduos durante o processo de cooperação pode resultar em indivíduos factíveis ou infactíveis. Um indivíduo é considerado factível quando está dentro do espaço de soluções e infactível quando estiver fora desse espaço. Quando indivíduos infactíveis são gerados, estes devem ser tratados de forma a resultar indivíduos factíveis ou então serem descartados.

[Liepens e Potter, 1991] descrevem três estratégias que podem ser seguidas para que apenas indivíduos factíveis sejam gerados. Uma maneira simples é rejeitar o indivíduo infactível obtido durante o processo de cooperação. Outra abordagem é desenvolver na fase de combinação, procedimentos especializados que gerem somente indivíduos factíveis. Esta última abordagem é freqüentemente usada em procedimentos de reparação que transformam indivíduos infactíveis em factíveis. Na terceira abordagem, também chamada de fase de adaptação, aceita-se o indivíduo infactível, mas este é penalizado na função de avaliação, a qual mensura a qualidade de um indivíduo.

Na fase de atualização, usualmente os indivíduos mais adaptados são levados para a geração seguinte.

Muitos algoritmos evolutivos podem ser melhorados utilizando procedimentos de busca

local no processo de adaptação. Tais procedimentos tentam melhorar o valor de um indivíduo sem usar informações de outros indivíduos.

O uso de procedimentos de busca local em algoritmos evolutivos tem se mostrado muito eficiente em muitas aplicações. A utilização de uma população de indivíduos assegura uma exploração de uma grande parte do espaço de busca, enquanto que os algoritmos de busca local ajudam a determinar indivíduos com boa qualidade em regiões identificadas como promissoras [Toth e Vigo, 2002].

2.4 Estratégias de Intensificação

2.4.1 Reconexão por Caminhos

A técnica Reconexão por Caminhos (RC), conhecida como *Path Relinking* - PR, é uma estratégia de intensificação, proposta por [Glover, 1996], que explora trajetórias que conectam duas soluções elite. O termo soluções elite se refere às soluções de boa qualidade geradas anteriormente à aplicação da técnica ou durante sua aplicação.

Segundo [Rosseti, 2003], a técnica de Reconexão por Caminhos pode ser aplicada segundo duas estratégias:

- Como uma estratégia de pós-otimização entre todos os pares de soluções elite;
- Como uma estratégia de intensificação a cada ótimo local encontrado após a fase de busca local.

A técnica de Reconexão por Caminhos aplicada como uma estratégia de intensificação a cada ótimo local é aplicada a pares (s_1, s_2) de soluções, onde s_1 é a solução corrente encontrada após um processo de busca local e s_2 é uma solução elite escolhida de forma aleatória, pertencente a um conjunto limitado de soluções elite. Como o conjunto de soluções elite encontra-se inicialmente vazio, todas as soluções encontradas na fase de busca local são apenas inseridas no conjunto. Quando o conjunto de soluções elite estiver cheio, a cada busca local, a solução obtida torna-se candidata a fazer parte desse conjunto. Para uma solução entrar nesse conjunto é necessário que ela seja melhor que a pior solução do conjunto. Além disso, é necessário que a solução candidata possua um percentual mínimo de diferença para cada solução pertencente ao conjunto de soluções elite. Neste trabalho, supõe-se sempre que se está trabalhando com problemas de minimização.

O Algoritmo 8 ilustra o pseudocódigo da técnica Reconexão por Caminhos unidirecional.

Algoritmo 8 Reconexão por Caminhos

1: $\bar{g} \leftarrow s$;
2: Atribuir a g' a melhor solução entre s e g;
3: Calcular o conjunto de movimentos possíveis $\Delta(s, g)$;
4: **enquanto** $|\Delta(s, g)| \neq 0$ **faça**
5: Atribuir a g'' a melhor solução obtida aplicando o melhor movimento de $\Delta(s, g)$ a $\bar{g}$;
6: Excluir de $\Delta(s, g)$ este movimento;
7: $\bar{g} \leftarrow g''$;
8: **se** $f(\bar{g}) < f(g')$ **então**
9: $g' \leftarrow \bar{g}$;
10: **fim se**
11: **fim enquanto**
12: Retorne g';

O algoritmo inicia-se calculando a diferença simétrica $\Delta(s, g)$ cujo valor representa o número de movimentos que deve ser aplicado a s, dita solução base, para que se chegue à solução guia g. A cada iteração, o melhor movimento de $\Delta(s, g)$ ainda não aplicado à solução base é efetuado até que se alcance a solução guia. Após a aplicação do movimento, esse é excluído do conjunto e, finalmente, verifica-se se a solução base melhora a melhor solução encontrada até então. Caso isso ocorra, a solução base torna-se candidata a se tornar uma solução elite. O algoritmo termina quando ocorre a convergência da solução base para a solução guia.

De acordo com [Rosseti, 2003], diversas sugestões podem ser consideradas e combinadas:

- Não aplicar a Reconexão por Caminhos a cada iteração, mas sim periodicamente, para não onerar o tempo final do algoritmo;
- Explorar duas trajetórias potencialmente diferentes, usando s_1 como solução base e depois s_2 no mesmo papel (Reconexão por Caminhos Bidirecional);
- Explorar apenas uma trajetória, usando s_1 ou s_2 como solução base;
- Não percorrer a trajetória completa de s_1 até s_2, mas sim apenas parte dela (Reconexão por Caminhos Truncada).

Para a escolha da alternativa mais apropriada, deve-se ter um compromisso entre tempo de processamento e qualidade da solução. Em [Ribeiro *et al.*, 2002] foi observado

que a exploração de duas trajetórias potencialmente diferentes para cada par (s_1, s_2) de soluções consome, aproximadamente, o dobro do tempo de processamento necessário para explorar apenas uma delas, com um ganho marginal muito pequeno em termos de qualidade de solução. Também foi observado pelos autores que, se apenas uma trajetória deve ser investigada, melhores soluções tendem a ser obtidas quando a Reconexão por Caminhos usa como solução inicial a melhor solução dentre s_1 e s_2 (esta é a chamada Reconexão por Caminhos Regressiva - *Backward Path Relinking*). Como a vizinhança da solução inicial é explorada com muito mais profundidade do que aquela da solução guia, usar como solução inicial a melhor dentre s_1 e s_2 oferece mais chances ao algoritmo de investigar mais detalhadamente a vizinhança da solução de melhor qualidade. Pela mesma razão, as melhores soluções são normalmente encontradas próximas da solução inicial, permitindo que o procedimento de Reconexão por Caminhos seja interrompido após algumas iterações, antes de a solução guia ser alcançada [Souza, 2008].

Capítulo 3

O Problema de Recobrimento de Rotas abordado

3.1 Descrição do Problema abordado

Este capítulo descreve de forma concisa o Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP), apresenta os trabalhos relacionados ao problema como também alguns problemas similares e exemplifica algumas aplicações do PRRCP em situações reais.

O Problema de Recobrimento de Rotas com Coleta de Prêmios - PRRCP (*The Prize Collecting Covering Tour Problem* - PCCTP), é uma variação do Problema de Recobrimento de Rotas - PRR (*The Covering Tour Problem* - CTP), que por sua vez é uma generalização do Problema do Caixeiro Viajante (PCV).

O PRR pode ser definido em um grafo simétrico, não-direcionado $G = (N, E)$, em que:

- $N = V \cup W$ representa o conjunto de vértices, onde $V \cap W = \emptyset$;
- $T \subseteq V$ é o conjunto de vértices obrigatórios, isto é, que *devem* fazer parte da rota;
- $V \setminus T$ é o conjunto de vértices *opcionais*, isto é, não necessariamente precisam fazer parte da rota;
- $V = T \cup (V \setminus T)$ representa o conjunto de vértices que *podem* fazer parte da rota;

- W é formado pelos vértices que precisam ser cobertos por algum vértice pertencente a V, isto é, deve existir um vértice na rota pertencente a V que esteja a uma distância igual ou inferior a D de um vértice W, onde D é um dado de entrada do problema;
- Os vértices pertencentes a W não podem estar presentes na rota.
- $E = \{(i, j) \mid i, j \in N\}$ representa o conjunto de arestas;
- Existe uma matriz simétrica de distâncias c_{ij}, com $i, j \in N$, definida sobre o subconjunto de arestas E, satisfazendo a desigualdade triangular.

A partir destas considerações, tem-se que o PRR consiste em determinar uma rota ou ciclo Hamiltoniano de comprimento mínimo sobre um subconjunto de V, contendo todos os vértices obrigatórios de T e cobrindo todos os vértices de W.

Um exemplo de solução para o PRR está ilustrado na Figura 3.1. Nesta figura, os círculos representam os vértices obrigatórios de T, os triângulos são os vértices de $V \backslash T$ e os pentágonos são aqueles que precisam ser cobertos. A área delimitada pela circunferência de raio D em cada vértice W corresponde à área na qual deve existir pelo menos um vértice de T ou $V \setminus T$ para que o vértice de W seja coberto. Note que, para cada $j \in W$, qualquer vértice $i \in V$, tal que $c_{ij} \leq D$, pode ser utilizado na rota para cobrir j. As arestas do grafo, com exceção das que fazem parte da solução, não foram ilustradas para facilitar a visualização.

O Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP), além das restrições comuns ao PRR, possui as seguintes particularidades:

- A cada vértice i de V está associado um prêmio não-negativo p_i;
- $\sum_{i \in s} p_i \geq PRIZE$, onde s é a rota solução e $PRIZE$ é um prêmio mínimo pré-estabelecido que deve ser coletado.

O objetivo do PRRCP, assim como no PRR, é encontrar uma rota de comprimento mínimo em um subconjunto de V, satisfazendo ao seguinte conjunto de restrições:

- A rota deve conter todos os vértices de $T \in V$;
- Todos os vértices de W devem estar cobertos.

No PRRCP, tem-se a restrição adicional:

- Deve-se coletar, pelo menos, a quantidade mínima de prêmio, dada por *PRIZE*.

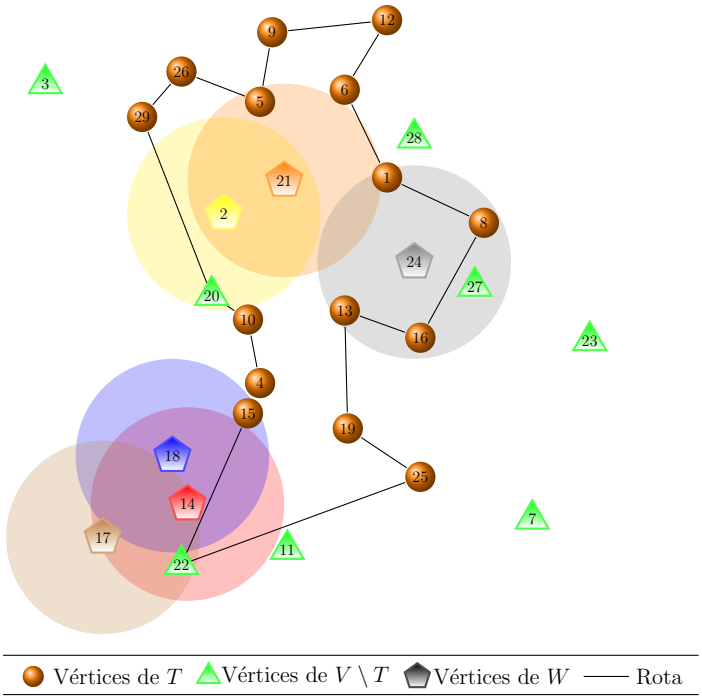

Figura 3.1: Grafo associado a uma instância do PRR.

De acordo com a Figura 3.1, verifica-se que foi necessário a inclusão de dois vértices opcionais na rota, pois os vértices 2 e 17 só eram cobertos pelos vértices 20 e 22 respectivamente.

Uma vez que o PRR pode ser reduzido ao Problema do Caixeiro Viajante (PCV) fazendo-se $D = 0$, $W = \emptyset$ e $V = T$, e o PCV é NP-difícil, então o PRR também o é.

A Figura 3.2 apresenta um exemplo de uma solução para o PRRCP, no qual o prêmio mínimo a ser coletado é 100. Observe que o conjunto de vértices desta figura é o mesmo da Figura 3.1, diferenciando-se desta apenas nos prêmios que foram inseridos nos vértices dos conjuntos T e $V \setminus T$. A área delimitada pela circunferência de raio D em cada vértice de W corresponde à área na qual deve existir pelo menos um vértice de T ou de $V \setminus T$ para que o vértice de W seja coberto. Note que, para cada $j \in W$, qualquer vértice $i \in V$, tal que $c_{ij} \leq D$, pode ser utilizado na rota para cobrir j.

Pela Figura 3.2, verifica-se que os vértices 3, 11, 23 e 28, mesmo não cobrindo nenhum vértice de W, fazem parte da rota pela necessidade de se coletar o prêmio mínimo.

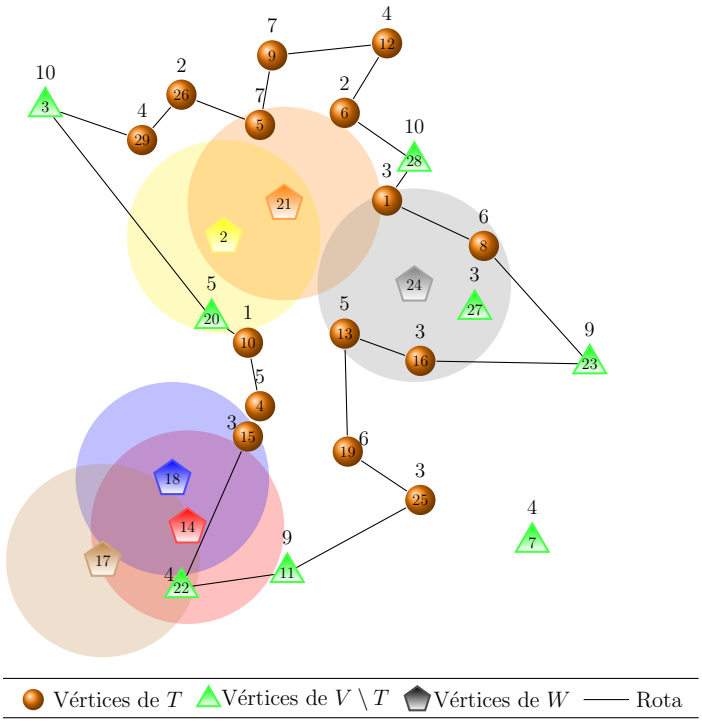

Figura 3.2: Grafo associado a uma instância do PRRCP.

Com base nas figuras anteriores, pode-se concluir que a inclusão da restrição de coleta de prêmio no PRR torna elegíveis a fazer parte da rota, também os vértices pertencentes ao conjunto $V \setminus T$ que não cobrem nenhum vértice de W. Isto se deve ao fato de que no PRR tais vértices (que não cobrem nenhum vértice do conjunto W) podem ser descartados mas, no PRRCP, por possuírem um prêmio p_i, são possíveis candidatos a fazer parte da rota no PRRCP.

Analogamente ao PRR, o PRRCP também é considerado um problema NP-Difícil, uma vez que pode ser reduzido ao PCV fazendo-se $D = 0$, $W = \emptyset$, $N = T$ e $PRIZE = 0$.

3.2 Problemas Similares

Existem diversos problemas na literatura similares ao PRR. Dentre eles, podem ser citados o *Shortest Covering Path Problem* (SCPP) [Current e Rolland, 1994, Current *et al.*, 1984], o *Multi-Vehicle Covering Tour Problem* (m-CTP) [Hachida *et al.*, 2000], o *Median Tour Problem* (MTP), o *Maximal Covering Tour Problem* (MCTP) [Current e Schilling, 1994], o *Prize Collecting Traveling Salesman Problem* (PCTSP) [Balas, 1989], o *Selective Traveling Salesman Problem* (STSP) [Laporte e Martello, 1990] e o *Geometric Covering Salesman Problem* (GCSP) [Arkin e Hassin, 1994].

Como exemplos de possíveis aplicações do PRR podemos mencionar o roteamento de aeronaves em vôos noturnos, onde a rota não inclui a visita em todas as cidades diretamente, e o planejamento de paradas de um circo durante uma estação. Esse último problema, conhecido como *Traveling Circus Problem* [Revelle e Laporte, 1993], consiste em encontrar uma rota onde as localidades não incluídas no planejamento de paradas estejam acessíveis a pelo menos uma das incluídas. Para cada localidade não visitada, acrescenta-se uma penalidade.

Outra possível aplicação do PRRCP consiste nos "Serviços de Atendimento Médico Móvel" (SAMM), que funcionam da seguinte forma: um veículo sai de um ponto origem, visita um conjunto de pontos de atendimento (vértices de T e eventualmente alguns vértices do conjunto $V \setminus T$), de forma que a população de uma determinada região (conjunto W) seja contemplada com tais serviços sem que nenhuma pessoa tenha que se locomover mais que uma distancia máxima D para chegar a um ponto de parada do veículo. Ao final, o veículo retorna ao local de origem. Os pontos de parada do veículo são definidos a partir de valores como: i) pontos de parada pré-estabelecidos (vértices de T), ii) pontos onde a quantidade de pessoas residentes na proximidade seja alta (prêmio p_i do vértice $i \in V$), e/ou que não estejam cobertas por pontos de T (vértices de $V \setminus T$).

Capítulo 4

Formulações Matemáticas para o PRRCP

Neste capítulo são apresentadas duas formulações matemáticas para resolução do PRRCP. A primeira é a única encontrada na literatura, foi proposta por [Lyra, 2004] e utiliza variáveis de fluxo para evitar ciclos desconexos da origem. A outra formulação, proposta no presente trabalho, utiliza variáveis de fluxo multi-*commodity* para evitar subciclos.

4.1 Formulação de Lyra

Segundo [Lyra, 2004], o PRRCP pode ser formulado como um modelo de Programação Linear Inteira (PLI) com as seguintes variáveis:

- Para toda aresta $(i, j) \in E$, seja z_{ij} com $i < j$, uma variável inteira não-negativa que representa a quantidade de fluxo escoado pela aresta (i, j);
- Para todo vértice $k \in V$, seja y_k uma variável binária que indica se o vértice k está presente ou não na solução;
- Para todo vértice $i, j \in V$, com $i < j$, seja x_{ij} uma variável binária que indica se a aresta $(i, j) \in E$ está presente na solução.

Além das variáveis apresentadas, seja c_{ij} o custo de se utilizar a aresta (i, j); p_k, o prêmio associado a cada vértice $k \in V$, *PRIZE*, o prêmio mínimo a ser coletado na rota e s o vértice origem.

De acordo com essas considerações, o PRRCP definido como um PPLI é descrito da seguinte forma:

$$\text{(P4)} \qquad \text{Minimizar} \sum_{i<j \,|\, i,j \in V} c_{ij} x_{ij} \tag{4.1}$$

$$\text{s.a:} \qquad \sum_{j \in V} p_k y_k \geq PRIZE \tag{4.2}$$

$$\sum_{k \in S_l} y_k \geq 1 \qquad \forall l \in W \tag{4.3}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2y_k \qquad \forall k \in V \tag{4.4}$$

$$\sum_{j \in V} z_{kj} = \sum_{i \in V} x_{ik} + 2y_k \qquad \forall k \in V \setminus \{s\} \tag{4.5}$$

$$\sum_{j \in V} z_{sj} = 1 \tag{4.6}$$

$$\sum_{j \in V} z_{js} = \sum_{j \in V} y_j \tag{4.7}$$

$$x_{ij} \leq z_{ij} \qquad \forall i, j \in V \tag{4.8}$$

$$x_{ij} \geq (z_{ij})/(|V| + |W| + 1) \qquad \forall i, j \in V \tag{4.9}$$

$$y_k = 1 \qquad k \in T \tag{4.10}$$

$$y_k \in \{0, 1\} \qquad k \in (V \setminus T) \tag{4.11}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{4.12}$$

$$z_{ij} \in Z^+ \qquad \forall i, j \in V \tag{4.13}$$

Nesta formulação, a função objetivo (4.1) visa a minimização do custo total da rota. A restrição (4.2) garante que o prêmio mínimo, *PRIZE*, será coletado. As restrições (4.3), nas quais tem-se que $S_l = \{i \in V : d_{il} \leq D, l \in W\}$, asseguram a cobertura de todos os vértices de W, garantindo que pelo menos um vértice $i \in V$, tal que $d_{il} \leq D$, esteja na rota. As restrições (4.4) são responsáveis pela conservação de fluxo, enquanto as restrições (4.5) a (4.7) asseguram a conectividade da rota. As restrições (4.8) e (4.9) garantem que a rota gerada a partir da variável de fluxo (z_{ij}) e a variável de decisão (x_{ij}) coincidam. As restrições (4.10) asseguram que todos vértices de T farão parte da rota. Por fim, as restrições (4.11) a (4.13) representam as condições de integralidade do problema.

De acordo com [Lyra, 2004], o número de variáveis e restrições dessa formulação é da ordem de $O(n^2)$ e $O(2m + p)$, respectivamente, sendo $n = |V \cup W|$, $p = |V|$ e $m = |E|$. Note que, para um grafo completo, $m = \frac{n(n-1)}{2}$.

4.2 Nova Formulação para o PRRCP

Propõe-se, nesta seção, um novo modelo de Programação Linear Inteira (PLI) para o PRRCP. Nesta formulação, variáveis de fluxo multiproduto são introduzidas com o intuito de evitar ciclos desconexos da origem. A formulação que utiliza esse tipo de variável foi introduzida por [Wong, 1980] e [Claus, 1984] para resolução do TSP, sendo esta a base para a criação da formulação proposta para o PRRCP. Esse tipo de variável é utilizado quando se deseja escoar diferentes mercadorias (*commodities*) pela mesma rede. Para facilitar o entendimento da formulação proposta, suponha que cada vértice pertencente à solução possua a demanda de uma unidade de uma mercadoria qualquer. Desta forma, o que se deseja é entregar cada mercadoria ao seu destino com o menor custo do percurso.

Sejam as seguintes variáveis:

- x_{ij}: variável binária que assume valor 1 se a aresta $(i, j) \in E$ está presente na solução e 0, caso contrário.
- z_k: variável binária que assume valor 1 se o vértice $k \in V$ está presente na solução e 0, caso contrário. Se $k \in T$, então $z_k = 1$;
- y_{ij}^k: varíavel inteira não-negativa que representa a quantidade de fluxo do produto k escoado pela aresta $(i, j) \in E$.

Além das variáveis apresentadas, seja c_{ij} o custo de se utilizar a aresta $(i, j) \in E$, p_i o prêmio associado a cada vértice $i \in V$ e *PRIZE*, o prêmio mínimo a ser coletado na rota.

De posse dessas considerações, propõe-se a seguinte formulação para o PRRCP:

$$\text{(P5)} \qquad \text{Minimizar} \sum_{\substack{i,j \in V \\ i \neq j}} c_{ij} x_{ij} \tag{4.14}$$

$$\text{s.a:} \qquad \sum_{\substack{j \in V \\ j \neq i}} x_{ij} = z_i \qquad \forall i \in V \tag{4.15}$$

$$\sum_{\substack{i \in V \\ i \neq j}} x_{ji} = z_j \qquad \forall j \in V \tag{4.16}$$

$$y_{ij}^k \leq x_{ij} \qquad \forall i, j, k \in V \tag{4.17}$$

$$\sum_{i \in V} y_{1i}^k = z_k \qquad \forall k \in V \setminus \{s\} \tag{4.18}$$

$$\sum_{i \in V} y_{i1}^k = 0 \qquad \forall k \in V \setminus \{s\} \tag{4.19}$$

$$\sum_{i \in V} y_{ik}^k = z_k \qquad \forall k \in V \setminus \{s\} \tag{4.20}$$

$$\sum_{j \in V} y_{kj}^k = 0 \qquad \forall k \in V \setminus \{s\} \tag{4.21}$$

$$\sum_{i \in V} y_{ij}^k - \sum_{i \in V} y_{ji}^k = 0 \qquad \forall k, j \in V \setminus \{s\}, j \neq k \tag{4.22}$$

$$\sum_{i \in V} p_i z_i \geq PRIZE \tag{4.23}$$

$$\sum_{i \in S_l} z_i \geq 1 \qquad \forall l \in W \tag{4.24}$$

$$z_i = 1 \qquad \forall i \in T \tag{4.25}$$

$$z_i \in \{0, 1\} \qquad \forall i \in (V \setminus T) \tag{4.26}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{4.27}$$

$$y_{ij}^k \in I^+ \qquad \forall i, j, k \in V \tag{4.28}$$

Nesta formulação, a função (4.14) tem por objetivo minimizar o custo total da rota. As restrições (4.15) e (4.16) são responsáveis pela conservação de fluxo dos vértices da solução e as restrições (4.17) garantem que somente passará fluxo de algum produto apenas nas arestas presentes na solução. As restrições (4.18) asseguram que será escoado um produto da origem para cada cliente que estiver na rota, enquanto as restrições (4.19) não permitem que nenhum produto retorne à origem. As restrições (4.20) e (4.21) impõem que todo vértice da solução receberá o seu produto correspondente e que esse, ao chegar no seu destino final (vértice correspondente), não será escoado para nenhum outro vértice. As restrições (4.22) garantem a conservação de fluxo dos produtos que não estiverem alcançados os seus vértices de destino. A restrição (4.23) garante que o prêmio mínimo, *PRIZE*, será coletado. As restrições (4.24), nas quais têm-se que $S_l = \{i \in V : d_{il} \leq D,\ l \in W\}$, asseguram a cobertura de todos os vértices de W, garantindo que pelo menos um vértice $i \in V$, tal que $d_{il} \leq D$, esteja na rota. As restrições (4.25) asseguram que todos os vértices de T farão parte da rota. Por fim, as restrições (4.26) a (4.28) representam as condições de integralidade das variáveis z_i, x_{ij} e y_{ij}^k.

De acordo com [Wong, 1980] e [Claus, 1984], o número de restrições e variáveis desta formulação é da ordem de $O(n^3)$ e $O(n^3)$, respectivamente. A grande vantagem em se utilizar esse tipo de variáveis multiproduto está no número reduzido de nós que os resolvedores matemáticos precisam explorar na árvore de *Branch-and-Bound* [Orman e Williams, 2005].

Capítulo 5

Regras de Redução para o PRR e o PRRCP

Este capítulo apresenta as regras de redução existentes na literatura para o PRR e o PRRCP, bem como propõe novas regras para o PRRCP. O objetivo de se utilizar essas regras é diminuir o número de vértices do problema e, conseqüentemente, o número de arestas do grafo, de forma a reduzir o número de combinações a serem analisadas, melhorando-se, assim, o esforço computacional da exploração do espaço de busca. Obviamente, na aplicação das regras, deve-se ter o cuidado para não excluir nenhum vértice que possa fazer parte de uma solução ótima.

5.1 Regras de Redução para o PRR

Descreve-se, a seguir, regras de redução para o PRR, introduzidas por [Gendreau *et al.*, 1995].

Considere (T), $(V \setminus T)$ e (W), os conjuntos de vértices do grafo associado ao PRR, definidos anteriormente e ξ_{ij}, um parâmetro que representa a condição de cobertura do vértice $i \in W$ em relação ao vértice $j \in V$, onde $\xi_{ij} = 1$ se o vértice i é coberto pelo vértice j e 0, caso contrário. O conjunto de regras de redução apresentadas em [Gendreau *et al.*, 1995] é descrito por:

1. Remover $i \in W$, se $\forall j \in V \setminus T$, $\xi_{ij} = 1$;
2. Remover $i \in W$, se existir $j \neq i$, com $j \in W$, tal que $\xi_{ik} \leq \xi_{jk}$, $\forall k \in V \setminus T$;
3. Remover $i \in W$, se existir $j \in T$, tal que $\xi_{ij} = 1$;
4. Remover $i \in V \setminus T$, se $\forall j \in W$, $\xi_{ij} = 0$.

As regras **(3)** e **(4)** se aplicam perfeitamente ao PRR. Entretanto, em sua dissertação, [Motta, 2001] mostra que as regras **(1)** e **(2)** podem gerar soluções inviáveis para o PRR.

Em relação ao PRRCP, somente a regra **(3)** pode ser utilizada da mesma maneira que no PRR. Em seu trabalho, [Lyra, 2004] adapta as regras **(1)** e **(4)** e propõe uma nova regra de redução para o PRRCP. No presente trabalho, apresentam-se três novas regras de redução para o PRRCP.

5.1.1 Análise das Regras de Redução associadas ao PRR

Esta seção analisa e verifica a aplicabilidade de cada uma das regras existentes na literatura para redução do grafo associado ao PRR.

Considere o grafo de uma instância do PRR ilustrado na Figura 5.1. Neste grafo, os vértices $i \in W$ podem ser cobertos por qualquer vértice opcional $j \in V \setminus T$. Portanto, uma solução viável para este caso pode ser obtida considerando uma rota que contenha todos os vértices de T juntamente com qualquer um dos vértices de $V \setminus T$.

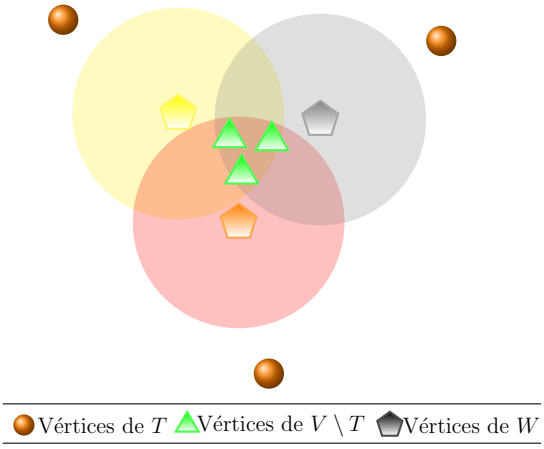

Figura 5.1: Grafo associado a uma instância do PRR para análise da regra (1).

Se a regra de redução **(1)** for aplicada ao grafo da Figura 5.1, todos os vértices de W são removidos (Figura 5.2). Com isso, pode-se reduzir o espaço de soluções considerando-se o grafo reduzido contendo apenas os vértices obrigatórios (Figura 5.3), já que W torna-se um conjunto vazio e a presença dos vértices optativos ($V \setminus T$) na rota não se justifica, uma vez que as distâncias entre os vértices são euclidianas.

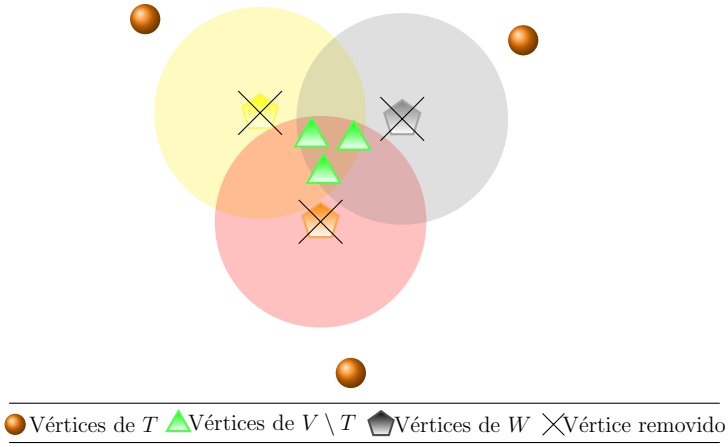

Figura 5.2: Redução do grafo da Figura 5.1, regra (1).

Contudo, a solução viável associada ao grafo reduzido (Figura 5.3) é inviável para o grafo original (Figura 5.1).

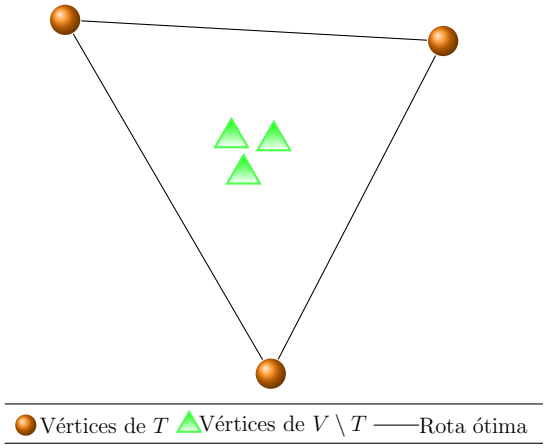

Figura 5.3: Grafo da Figura 5.1, após a redução, com a rota ótima associada.

A regra **(2)** elimina do grafo original todo vértice de $i \in W$, quando existir um outro vértice $j \neq i \in W$, tal que todo $k \in V \setminus T$ pertencente à vizinhança de i também pertença à vizinhança de j, mas nem todo $k \in V \setminus T$ pertencente à vizinhança de j pertença à vizinhança de i. Observe esta situação ilustrada na Figura 5.4. Se a redução for realizada de acordo com a regra **(2)**, um vértice de W será eliminado (Figura 5.5) e o grafo reduzido terá como solução ótima (Figura 5.6) uma solução inviável em relação ao grafo original.

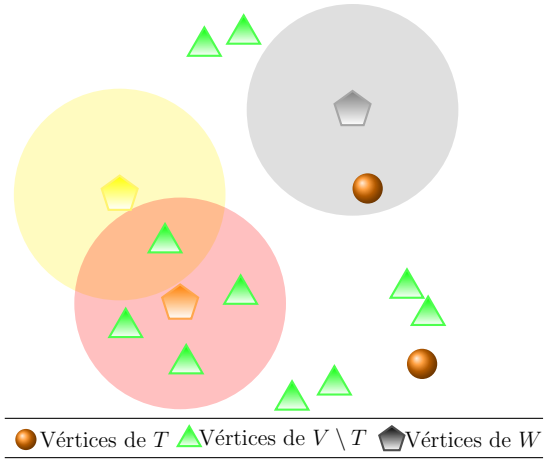

Figura 5.4: Grafo associado a uma instância do PRR para análise da regra (2).

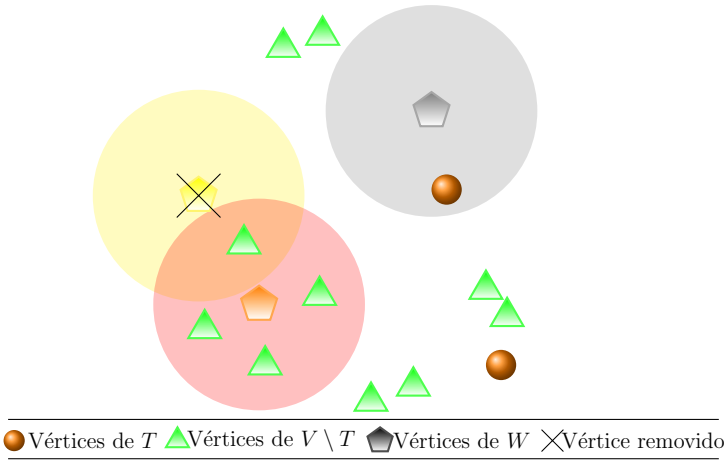

Figura 5.5: Resultado da aplicação da regra (2) de redução ao grafo da Figura 5.4.

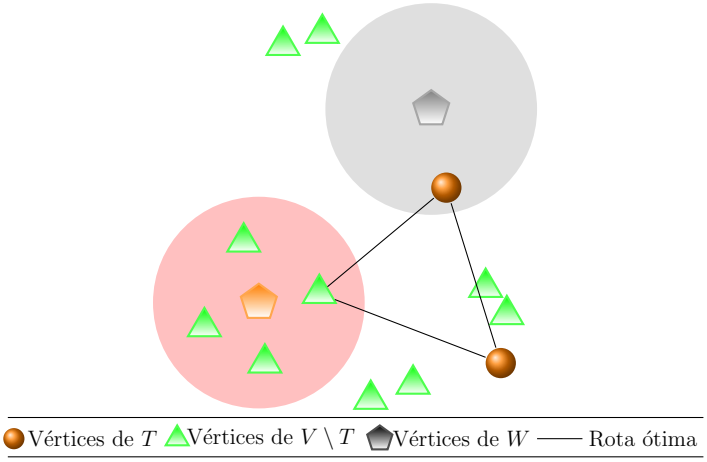

Figura 5.6: Resultado da aplicação da regra (2) ao grafo da Figura 5.4. A rota associada é inviável para o grafo original.

Com relação às regras **(3)** e **(4)**, estas se aplicam perfeitamente ao PRR. Observe novamente o grafo original apresentado na Figura 5.4. A Figura 5.7 mostra os vértices que seriam eliminados com a aplicação da regra **(3)**, que elimina do grafo original os vértices de W cobertos por algum vértice de T e a Figura 5.8 exemplifica a regra **(4)**, que elimina somente os vértices de $V \setminus T$ que não cobrem nenhum vértice de W. Por fim, a Figura 5.9 mostra como fica o grafo original após a aplicação dessas duas regras. Os números que estão acima de alguns vértices indicam por qual regra tal vértice foi excluído.

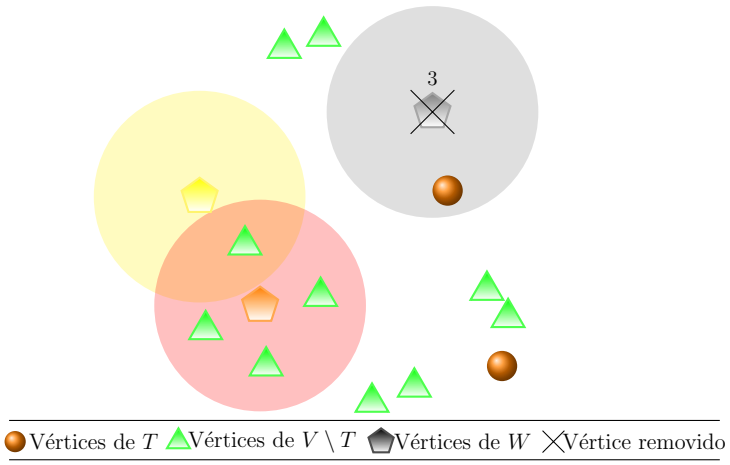

Figura 5.7: Resultado da aplicação da regra de redução (3) ao grafo da Figura 5.4.

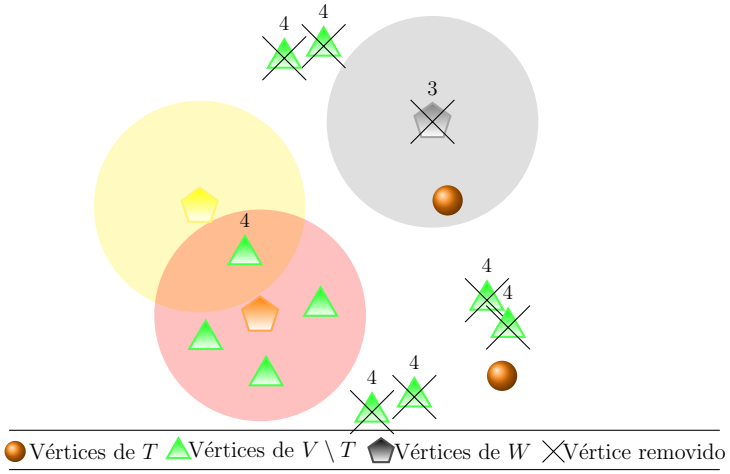

Figura 5.8: Resultado da aplicação da regra de redução (4) ao grafo da Figura 5.4.

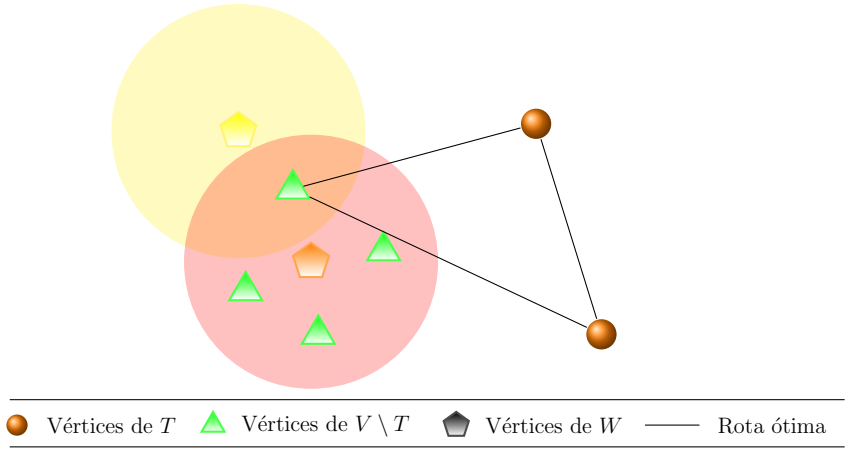

Figura 5.9: Grafo da Figura 5.4 após a redução pelas regras (3) e (4).

5.1.2 Análise das Regras de Redução associadas ao PRRCP

Esta seção mostra como adaptar o conjunto de regras existentes para o PRR ao PRRCP, apresenta a regra proposta por [Lyra, 2004], bem como as três novas regras propostas neste trabalho.

A regra **(1)**, após a adaptação apresentada em [Lyra, 2004] e a regra **(3)**, descritas anteriormente para o PRR, podem ser utilizadas sem correção no PRRCP, enquanto a regra **(4)** precisa ser modificada para que possa ser utilizada. Mostra-se também que, ao utilizar estas regras em uma determinada ordem, a possibilidade de se reduzir uma maior quantidade de vértices aumenta.

A grande diferença existente na redução do espaço de soluções do PRR e do PRRCP está na remoção dos vértices do conjunto $V \setminus T$. Remover um vértice de $V \setminus T$ no PRRCP consiste não somente em verificar sua condição de cobertura, mas também verificar se o prêmio mínimo pode ser coletado sem a necessidade de se utilizar tal vértice.

Seja $S_i = \{j \in V : d_{ij} \leq D, i \in W\}$, $Cob_j = \{i \in W : d_{ij} \leq D, j \in V\}$ e *flag* uma variável binária que recebe o valor 1 para indicar a necessidade de algum vértice de $V \setminus T$ fazer parte da solução. Essa variável foi introduzida para corrigir o problema da regra **(1)** para o PRR.

O conjunto e a ordem de aplicação das regras para redução do grafo associado ao PRRCP pode ser descrito da seguinte forma:

- **R1.** Para todo vértice $j \in V \setminus T$, para todo vértice $i \in W$, se $|S_i| = 1$ e $\xi_{ij} = 1$, transforme j em vértice obrigatório, isto é, $j \in T$; [Lyra, 2004]

- **R2.** Para todo vértice $j \in V \setminus T$, se ao removê-lo do conjunto a soma dos prêmios do restante dos vértices não atingir a coleta mínima, transforme j em vértice obrigatório; (Nova)

- **R3.** Se $\sum_{i \in T} p_i \geq PRIZE$, remover $j \in V \setminus T$, se $\forall i \in W$, $\xi_{ij} = 0$; [Lyra, 2004]

- **R4.** Se $\sum_{i \in T} p_i \geq PRIZE$, $\forall i \in W$, se existir um vértice $j \in V \setminus T$ e um vértice $k \in T$ tal que $\xi_{ij} = 1$ e $\xi_{ik} = 1$ e o vértice j somente cobrir o vértice i ou todos os vértices cobertos por j também são cobertos por um vértice de T, remover $j \in V \setminus T$; (Nova)

- **R5.** Para todo $i, j \in V \setminus T$, $i \neq j$, se $\sum_{k \in V-i} p_k \geq PRIZE$, $Cob_i \subseteq Cob_j$ e $d_{ik} > d_{jk}$, $\forall k \in V$, $k \neq i$, remover $i \in V \setminus T$; (Nova)

- **R6.** Remover $i \in W$, se existir $j \in T$, tal que $\xi_{ij} = 1$; [Gendreau *et al.*, 1995]

- **R7.** Remover $i \in W$, se $\forall j \in V \setminus T$, $\xi_{ij} = 1$ e atribuir o valor 1 à variável *flag*. [Lyra, 2004]

Para exemplificar a aplicação de cada uma das seis primeiras regras para redução do grafo associado ao PRRCP, seja a Figura 5.10. Nesta figura, suponha que o prêmio mínimo a ser coletado seja 100 ($PRIZE = 100$), e que o número acima de cada vértice pertencente ao conjunto V se refere ao prêmio que ele possui. A sétima regra será exemplificada separadamente por se tratar de um caso particular de configuração dos vértices em um grafo.

A Figura 5.11 apresenta o grafo associado a uma instância do PRRCP com todos os vértices que foram reduzidos com a aplicação das seis primeiras regras de redução.

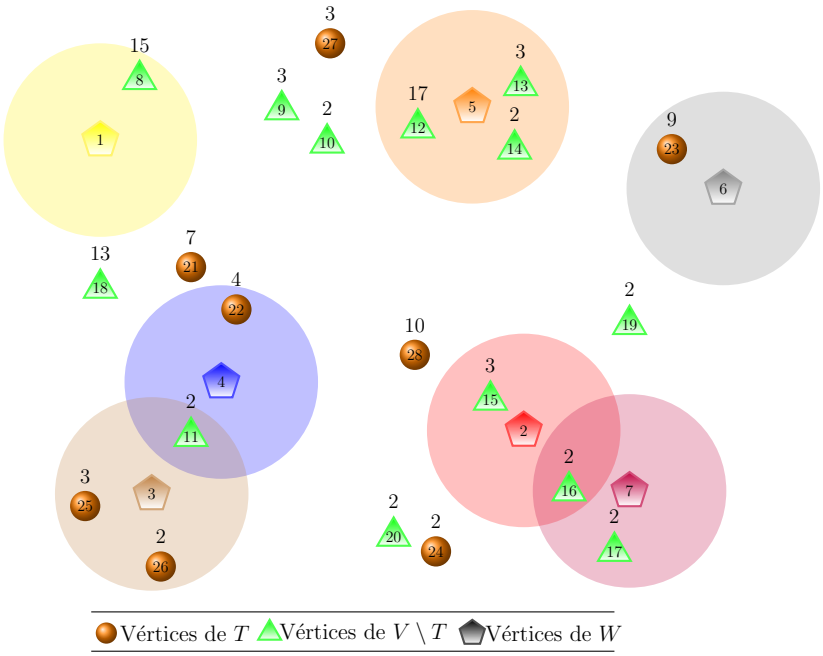

Figura 5.10: Grafo associado a uma instância do PRRCP.

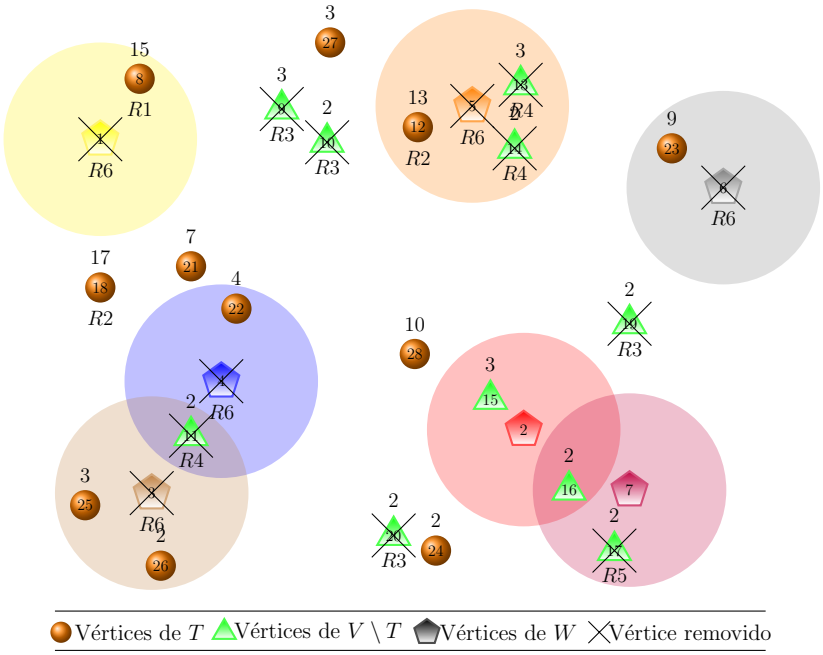

Figura 5.11: Resultado da aplicação da regra **R6** sobre o grafo da Figura 5.10.

A regra **R1**, proposta por [Lyra, 2004], transforma o vértice opcional ($j \in V \setminus T$) em obrigatório ($j \in T$), caso j seja o único vértice que cubra algum vértice $i \in W$. Observe o vértice 8 na Figura 5.10. Como ele é o único que cobre o vértice 1 e pertence ao conjunto de vértices opcionais, com a aplicação da regra, ele passa a pertencer ao conjunto de

vértices obrigatórios, como pode ser visto na Figura 5.11. A letra R com um número na frente, abaixo de alguns vértices, se refere à regra que se aplicou a tal vértice.

A regra **R2**, proposta neste trabalho, também transforma o vértice opcional ($j \in V \backslash T$) em obrigatório ($j \in T$) se, ao retirar o vértice j, o total de prêmio que pode ser coletado com os vértices que não foram removidos seja maior ou igual ao prêmio mínimo *PRIZE*. Observe na Figura 5.11 que os vértices opcionais 12 e 18, por possuírem um prêmio alto em relação aos demais vértices, com certeza estarão em qualquer solução viável para o PRRCP, pois sem esses vértices não se consegue coletar o prêmio mínimo. Dessa forma, com a aplicação da regra **R2**, os vértices 12 e 18 passam a pertencer ao conjunto T.

As regras **R1** e **R2** poderão reduzir a cardinalidade de $V \setminus T$, ajudando na aplicação da regra **R7**. Conseqüentemente, a cardinalidade de T aumentará, contribuindo para a remoção de vértices ao se aplicar as regras **R3**, **R4**, **R5** e **R6**. Isto se deve ao fato de que, ao transformar vértices opcionais em obrigatórios, aumenta-se a quantidade de prêmio coletado pelos vértices que, com certeza, estarão na solução ótima. O prêmio coletado apenas com esses vértices podem ser suficientes para atender ao prêmio mínimo. Quando isso acontece, pode-se tratar o PRRCP como uma instância do PRR, pois não existe mais a necessidade de se preocupar com a coleta de prêmios.

A regra **R3**, originalmente proposta para o PRR, precisou ser modificada para poder ser utilizada no PRRCP, pois cada vértice de V agora possui um prêmio associado. Na adaptação proposta, um vértice $k \in V \setminus T$ só será descartado caso este não seja necessário para se obter o prêmio mínimo (*PRIZE*) e se k não cobrir nenhum vértice de W. Observe na Figura 5.11 que os vértices opcionais 9, 10, 19 e 20 podem ser removidos, pois esses vértices não cobrem vértices de W e os obrigatórios já satisfazem ao prêmio mínimo.

A segunda regra proposta neste trabalho, a **R4**, diz que, para todo vértice ($j \in V \backslash T$), se o prêmio mínimo (*PRIZE*) puder ser atingido com a soma dos prêmios dos vértices obrigatórios, se j cobrir apenas um vértice de W ou se todos os vértices de W que j cobrir também forem cobertos por pelo menos um vértice de T, então pode-se remover j. A Figura 5.11 mostra que, com a aplicação da regra **R4**, os vértices 11, 13 e 14 são removidos.

A regra **R5**, também proposta no presente trabalho, mostra que, para quaisquer dois vértices ($i, j \in V \setminus T$, $i \neq j$), pode-se remover o vértice i, se as seguintes condições forem satisfeitas: (1) todos os vértices de W que i cobre também forem cobertos pelo vértice j; (2) a distância do vértice i para todos os outros vértices que podem fazem parte da rota (vértices pertencentes a V) é maior que a distância do vértice j para todos os mesmos

vértices de T e (3) o prêmio mínimo puder ser coletado com os prêmios dos vértices de V, com exceção de i.

Observe na Figura 5.11 que o vértice 17 pode ser eliminado pela regra **R5**, pois além da restrição do prêmio mínimo já estar satisfeita, o vértice 16 também cobre o vértice 7 e a distância do 16 para todos os vértices que podem fazer parte da solução é menor que a distância do 17 para esses mesmos vértices.

A regra **R6**, proposta para o PRR por [Gendreau *et al.*, 1995], pode ser aplicada sem modificações ao PRRCP. Ela remove todos os vértices de W que forem cobertos por algum vértice de T. Com a aplicação dessa regra, os vértices 1, 3, 4, 5 e 6 são removidos, como pode ser observado na Figura 5.11.

A Figura 5.12 apresenta uma possível solução para o grafo da Figura 5.10 após aplicação das seis primeiras regras de redução.

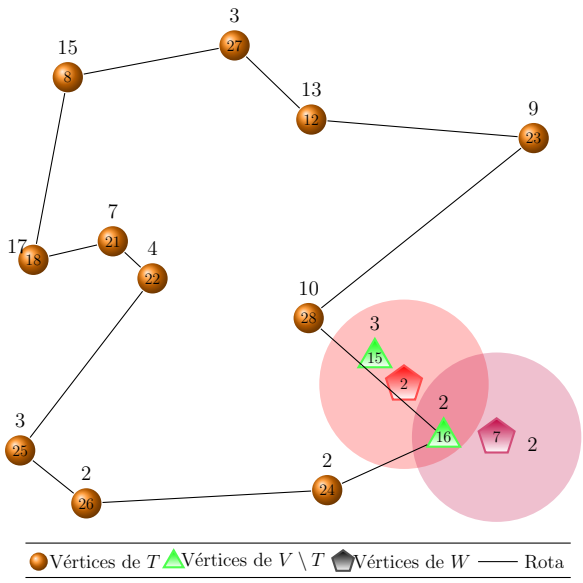

Figura 5.12: Possível solução para o grafo da Figura 5.10, após aplicação das seis primeiras regras de redução.

Por fim, a regra **R7**, proposta por [Lyra, 2004], consiste em remover um vértice ($i \in W$) se este for coberto por todos os vértices de $V \setminus T$. Essa regra, inicialmente proposta para o PRR, precisou ser modificada pois, caso todos os vértices de W sejam removidos, será necessário que pelo menos um vértice de $V \setminus T$ faça parte da solução, o que será informado pela variável auxiliar *flag*. A Figura 5.13 apresenta o resultado da regra **R7** ao grafo da Figura 5.10.

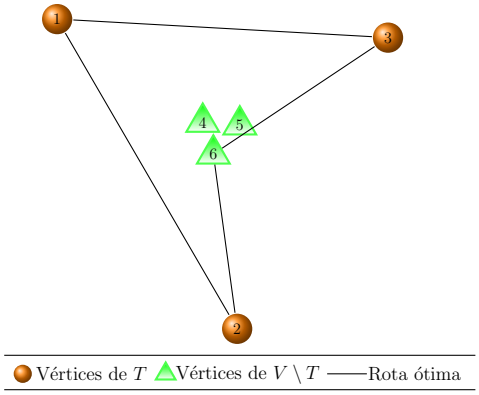

Figura 5.13: Grafo da Figura 5.1, após a redução pela regra **R7**, com a rota ótima viável associada.

A ordem proposta para a aplicação das regras de redução pode ser justificada levando-se em conta que a idéia dessas regras é primeiramente reduzir de alguma forma o conjunto de vértices que podem fazer parte da solução, seja excluindo-os ou mudando seu *status*, por exemplo, para vértice obrigatório. Então, para se obter o maior proveito dessas regras, é necessário primeiramente transformar em obrigatórios, os vértices opcionais que farão parte de qualquer solução viável, seja por ter um prêmio interessante ou por ser o único a cobrir algum vértice de W (Regras **R1** e **R2**). Como dito anteriormente, essas regras, se aplicadas, irão aumentar a cardinalidade do conjunto T e, conseqüentemente, aumentarão a quantidade de prêmio acumulado para se atingir a quantidade mínima de coleta. Se essa quantidade mínima de prêmio a ser coletado for obtida apenas nos vértices obrigatórios, as chances das regras **R3**, **R4** e **R5** serem aplicadas aumentam consideravelmente, pois além da restrição de prêmio mínimo já estar satisfeita, o aumento do número de vértices no conjunto T pode aumentar as chances de exclusão de vértices de $V \setminus T$, pois vértices que anteriormente eram cobertos por apenas vértices opcionais, podem agora ser cobertos por vértices opcionais e obrigatórios ou somente por obrigatórios. Por fim, após aplicar as regras que podem reduzir o conjunto de vértices potenciais a fazer parte da solução, a idéia é tentar excluir aqueles que precisam ser cobertos (Regras **R6** e **R7**). O objetivo de se excluir vértices de W consiste em atender uma restrição do PRRCP antes mesmo de se iniciar uma busca local. Isto é, o conjunto dos possíveis vértices a fazer parte da solução automaticamente já atendem a restrição de cobertura do conjunto W, o que pode ajudar os algoritmos propostos a utilizarem o tempo computacional de forma mais eficiente, pois se gastará mais tempo procurando a solução ótima do que tentando encontrar uma solução que seja apenas viável ao problema.

Capítulo 6

Heurísticas aplicadas ao PRRCP

Este capítulo descreve a metodologia utilizada para a resolução do Problema de Recobrimento de Rotas com Coleta de Prêmios. A Seção 6.1 mostra como uma solução é representada. A Seção 6.2 apresenta os algoritmos utilizados para geração de uma solução inicial para o problema. A Seção 6.3 exemplifica as estruturas de vizinhaça que foram utilizadas para explorar o espaço de soluções e a Seção 6.4 mostra como se avalia uma solução do PRRCP. Por fim, apresenta-se como as heurísticas e metaheurísticas descritas na Seção 2.2 e 2.3 foram aplicadas ao PRRCP.

6.1 Representação de uma solução para o PRRCP

Uma rota, ou solução s do problema, é representada por um vetor contendo no máximo $|T| + |V \setminus T|$ posições, pois como no PRRCP apenas um subconjunto dos vértices compõe a solução, o tamanho desse vetor pode variar de uma solução para outra.

Para exemplificar esse tipo de representação, seja a solução s apresentada no grafo da Figura 6.1.

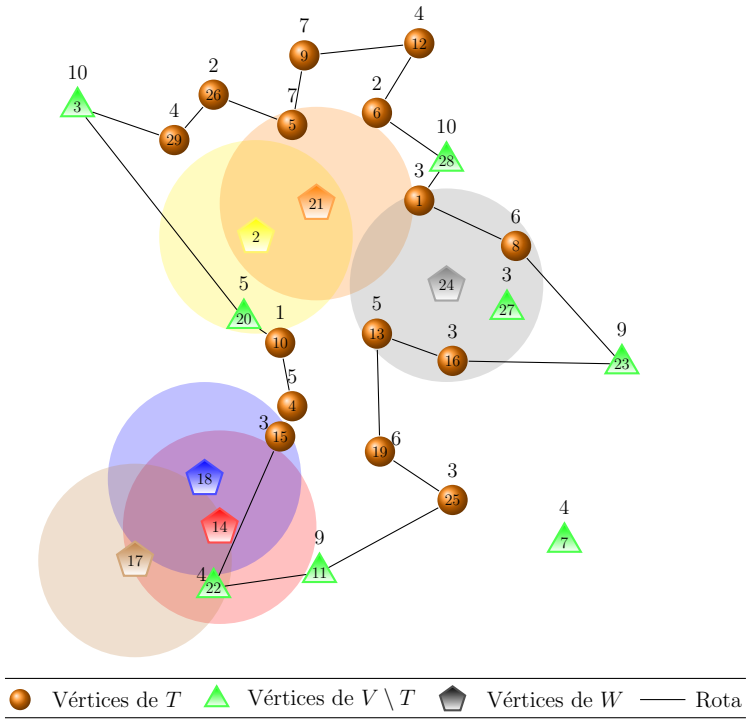

Figura 6.1: Grafo associado a uma instância do PRRCP.

A Figura 6.2 mostra a solução apresentada na Figura 6.1 na representação proposta neste trabalho.

13	19	25	11	22	15	4	10	20	3	29	26	5	9	12	6	28	1	8	23	16

Figura 6.2: Representação em vetor da solução apresentada na Figura 6.1.

Neste vetor, cada célula representa uma cidade, simbolizada por um número. A ordem dos números no vetor representa a ordem de visita das cidades. Como o problema consiste em definir uma rota, assume-se, para os cálculos que se fizerem necessários, que a última posição do vetor se conecta à primeira posição, formando um ciclo.

Para exemplificar a estrutura utilizada neste trabalho, tem-se na Figura 6.2 que, partindo-se da cidade 13, visita-se a cidade 19. Da cidade 19, visita-se a cidade 25 e assim sucessivamente. Para fechar a rota, da última cidade visitada (16) volta-se à cidade origem (13).

O tipo de representação da solução foi modificada, neste trabalho, para facilitar a visualização dos diferentes tipos de vértices (T e $V \setminus T$) que fazem parte da rota na solução. Para isso, utilizam-se diferentes formas geométricas para simbolizá-los. A Figura

6.3 apresenta a solução da Figura 6.2 utilizando-se formas geométricas.

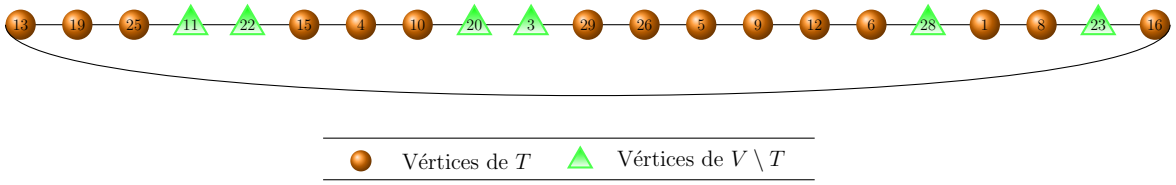

Figura 6.3: Representação através de formas geométricas da solução em vetor apresentada na Figura 6.2.

6.2 Métodos de geração de uma solução inicial para o PRRCP

Um fator importante a ser considerado, além da eficiência do algoritmo de busca, é a qualidade da solução inicial. Esta seção apresenta cinco algoritmos para a construção de uma solução inicial para o PRRCP. Cada um deles possui características distintas, permitindo-se avaliar quais deles melhor se adaptam ao problema tratado neste trabalho.

6.2.1 Algoritmo ADD

Este algoritmo baseia-se em um critério de inserção de vértices parcialmente guloso, tal como na fase de construção GRASP. A solução parcial inicia-se com um ciclo contendo dois vértices quaisquer do conjunto T (vértices obrigatórios). Seja LC a lista de vértices "disponíveis", isto é, a lista de vértices pertencentes ao conjunto V, que ainda não fazem parte da solução parcial e dessa forma, são candidatos a serem inseridos. A seguir, é formado um subconjunto dos vértices de LC, chamado de Lista Restrita de Candidatos (LRC), contendo os vértices que trazem um maior benefício para a solução com a sua inserção. Nesta lista, os vértices k são ordenados decrescentemente de acordo com a função de benefício, também chamada de economia, determinada pela função ρ, dada por:

$$\rho(k, i, j) = [c_{ij} - (c_{ik} + c_{kj})] + (M \times \phi) \tag{6.1}$$

em que:

- $\rho(k, i, j)$ fornece o benefício de se inserir o vértice k entre os vértices adjacentes i e

j, pertencentes à rota parcial;

- k representa o índice do vértice candidato à inserção;
- i e j são os índices dos vértices adjacentes na rota corrente entre os quais o vértice k será inserido;
- c_{ij}, c_{ik} e c_{kj} são as distâncias euclidianas entre os respectivos vértices;
- M é a soma de três parcelas: número de vértices de T não pertencentes à rota parcial, número de vértices de W não cobertos por ela e a quantidade de prêmio que falta ser coletada para satisfazer o prêmio mínimo $PRIZE$;
- ϕ é um valor inteiro que representa a maior distância entre dois vértices acrescido de uma unidade.

Vale ressaltar que as três parcelas que compõem M são calculadas partindo-se do pressuposto que k faça parte da rota.

Para um vértice k pertencer à LRC é preciso que $\rho(k,i,j) \geq \rho_{\max} - \alpha \times (\rho_{\max} - \rho_{\min})$, sendo $\rho_{\min}$ a menor economia de se inserir na solução parcial um vértice $k \in LC$ e $\rho_{\max}$, o vértice de maior economia. Tal como na Seção 2.3.1, o parâmetro α indica o nível de aleatoriedade da escolha.

A cada iteração desse algoritmo, um vértice k é selecionado aleatoriamente da LRC. Os vértices são inseridos na rota até que nenhuma economia positiva possa ser alcançada. Note que, de acordo com a função de economia ρ, os vértices são inseridos na rota até que uma solução viável seja encontrada, pois como o valor de ϕ é maior que a maior distância entre dois vértices do grafo, é sempre uma economia positiva no custo total da rota inserir novos vértices até que a viabilidade da solução seja atendida. O Algoritmo 9 aprensenta o pseudocódigo do Algoritmo ADD.

Algoritmo 9 Algoritmo *ADD*

1: $s \leftarrow \{i - j\}, i, j \in T$ (Selecionados aleatoriamente);
2: $f(s) \leftarrow N_W + N_T + N_P$;
3: $LC \leftarrow (V - \{i, j\})$;
4: **enquanto** $(f(s) > 0)$ **faça**
5: Definir LRC;
6: $k \leftarrow$ vértice selecionado aleatoriamente de LRC;
7: $pos \leftarrow$ melhor posição (i, j) para se inserir k na rota;
8: $s \leftarrow s \cup \{k\}$;
9: $LC \leftarrow LC \setminus \{k\}$;
10: $f(s) \leftarrow N_W + N_T + N_P$;
11: **fim enquanto**
12: Retorne s;

O algoritmo começa criando a rota parcial inicial formada pela escolha aleatória de dois vértices do conjunto T. A partir dessa rota parcial, inicializa-se a função de custo da rota com um valor equivalente ao produto de uma penalidade pela soma de três fatores: (i) quantidade de vértices de W não cobertos (determinado por N_W, sendo $|W| \geq N_W \geq 0$), (ii) quantidade de vértices obrigatórios não pertencentes à rota parcial (N_T, tal que $|T| \geq N_T \geq 0$) e (iii) quantidade de prêmio que ainda falta para satisfazer ao prêmio mínimo (N_P). O valor dessa penalidade, chamada de *Multa*, é definido com base na maior distância possível entre quaisquer dois vértices do grafo, visando a garantir a viabilidade das soluções geradas pelo algoritmo. É importante lembrar que a viabilidade está associada à cobertura de todo o conjunto W, ao fato de que todos os vértices obrigatórios devem pertencer à solução e que o prêmio mínimo pré-estabelecido deve ser coletado. A próxima etapa consiste em inicializar a lista de vértices candidatos a fazer parte da solução (LC), isto é, os vértices do grafo pertencentes ao conjunto V que não pertencem à s. Enquanto existir economia positiva na inserção de vértices em s, o algoritmo prossegue definindo a LRC e escolhendo um vértice deste conjunto de forma aleatória para ser inserido na rota. Vale ressaltar que, como existe a restrição de que todos os vértices de T precisam fazer parte da solução, é dada uma prioridade para que esses vértices sejam escolhidos. A variável *pos* armazena o par ordenado (i, j) de vértices i e j adjacentes entre o qual k será inserido. Ao definir a posição de inserção do vértice k, insere-se k em s, atualiza-se a lista de candidatos (LC) e a função de custo.

6.2.2 Algoritmo DROP

Este procedimento obtém uma solução inicial para o PRRCP utilizando um critério de remoção de vértices também parcialmente guloso. O ponto de partida deste algoritmo

é fornecido por uma rota associada à uma solução que contenha todos os vértices de V. Para se obter uma solução contendo todos os vértices de V, utiliza-se o Algoritmo ADD (Seção 6.2.1), diferindo-se deste apenas no critério de parada que, neste caso, considera a inserção dos vértices até que o conjunto LC fique vazio. De posse da solução inicial, um processo de remoções sucessivas é então realizado. A seleção dos vértices a serem removidos é feita a partir da LRC, um subconjunto de LC, que neste caso contém todos os vértices de $V \setminus T$ da rota corrente. Os vértices pertencentes à LRC são os que trazem as maiores contribuições caso sejam removidos da solução e são calculados de forma análoga ao Algoritmo ADD. A contribuição da remoção de cada vértice k é calculada pela função γ, definida por:

$$\gamma(k) = [(c_{ik} + c_{kj}) - c_{ij}] - (M \times \phi) + slackPRIZE \tag{6.2}$$

em que:

- k representa o índice do vértice candidato à remoção;
- $\gamma(k)$ fornece o benefício de se remover o vértice k da solução parcial;
- i e j são os índices dos vértices adjacentes ao vértice k;
- c_{ij}, c_{ik} e c_{kj} são as distâncias euclidianas entre os respectivos vértices;
- M é o número de vértices de W não cobertos na rota;
- ϕ é um valor inteiro que representa a maior distância entre dois vértices;
- *slackPRIZE* é a diferença entre o prêmio acumulado da rota ao se retirar o prêmio associado ao vértice k e o prêmio que deve ser coletado ($PRIZE$).

Os vértices são selecionados para a remoção pela ordem decrescente da economia associada à sua remoção. Enquanto for possível obter uma economia positiva sem perder a viabilidade, um novo vértice é escolhido para ser removido da rota corrente. Note que, como a rota contém inicialmente todos os vértices do grafo, acabam sendo removidos aqueles que acarretam os maiores acréscimos na distância total percorrida pela rota. O Algoritmo 10 descreve o pseudocódigo do Algoritmo DROP.

Algoritmo 10 Algoritmo *DROP*

1: $s \leftarrow$ Algoritmo ADD (Incluindo todos os vértices de V);
2: $f(s) \leftarrow$ custo da rota corrente;
3: $LC \leftarrow (V \setminus T)$;
4: **enquanto** ($M == 0$ && $slackPRIZE > 0$ && $f(s) > 0$) **faça**
5: Definir LRC;
6: $k \leftarrow$ vértice selecionado aleatoriamente de LRC;
7: $s \leftarrow s \setminus \{k\}$;
8: $LC \leftarrow LC \setminus \{k\}$;
9: $f(s) \leftarrow f(s) + [(c_{ik} + c_{kj}) - c_{ij}]$;
10: **fim enquanto**
11: Retorne s;

O algoritmo começa construindo uma solução inicial com todos os vértices de V, utilizando-se o Algoritmo ADD, apresentado na Seção 6.2.1. Após a criação da solução inicial, a lista de vértices candidatos (LC) a serem removidos dessa solução é formada inserindo-se todos os vértices que podem sair da solução, isto é, os vértices opcionais ($V \setminus T$). Enquanto existir economia positiva na remoção de vértices da solução, o algoritmo define os candidatos que trazem as maiores contribuições com a sua remoção e os colocam na LRC. A seguir, escolhe-se aleatoriamente um deles dessa lista para sair da rota. Escolhido o vértice, atualiza-se a solução, a lista de candidatos e a função de economia.

6.2.3 Algoritmo da Inserção Mais Barata

O Algoritmo da Inserção Mais Barata consiste em, partindo-se de uma rota parcial contendo dois vértices escolhidos de forma aleatória, construir passo a passo o restante da rota adicionando a cada passo, o vértice k que trouxer o menor custo adicional à solução. Para determinar qual vértice trará o menor custo à solução e a posição em que este deve ser inserido, avalia-se o custo de inserção de todos os possíveis vértices a fazer parte da solução entre todos os pares de vértices adjacentes que já estiverem na rota. O cálculo do custo de inserção de um vértice k entre dois vértices adjacentes i e j, dado por S_{ij}^k, é dado pela soma das distâncias das duas arestas que entrarão na solução, referentes às ligações entre os vértices i e k e os vértices k e j, subtraído da distância associada à aresta (i, j) que será removida da rota, isto é, $S_{ij}^k = c_{ik} + c_{kj} - c_{ij}$.

O Algoritmo 11 mostra o pseudocódigo do Algoritmo da Inserção Mais Barata.

Algoritmo 11 Algoritmo da Inserção Mais Barata

1: $s \leftarrow \{i - j\}, i, j \in T$ (Selecionados aleatoriamente);
2: $LC \leftarrow (V - \{i, j\})$;
3: **enquanto** $((\exists i \in W \textit{ não coberto}) \mid\mid (\exists j \in T \notin s) \mid\mid (\sum_{i \in s} p_i < PRIZE))$ **faça**
4: Defina o custo de inserção dos vértice de LC entre quaisquer pares de vértices de s;
5: Defina LRC;
6: $k \leftarrow$ vértice selecionado aleatoriamente de LRC;
7: $pos \leftarrow$ melhor posição (i, j) para se inserir k na rota;
8: $s \leftarrow s \cup k$;
9: $LC \leftarrow LC \setminus k$;
10: **fim enquanto**
11: Retorne s;

O algoritmo inicia-se construindo uma rota parcial com dois vértices pertencentes ao conjunto T, escolhidos aleatoriamente. Os vértices candidatos a entrarem na solução (vértices de T que ainda não foram inseridos na solução e os vértices de $V \setminus T$) são inseridos na lista de candidatos (LC). Construída a LC, enquanto todos os vértices de W não estiverem cobertos por algum vértice da solução, enquanto existir algum vértice de T fora da solução e o prêmio mínimo não estiver sido coletado, o algoritmo calcula o custo de inserção de todos os vértices ($k \in LC$) em qualquer posição entre dois vértices adjacentes. Os vértices k que possuírem o custo $S_{ij}^k \leq S_{\min} + \alpha \times (S_{\max} - S_{\min})$ são inseridos em LRC, sendo $S_{\min}$ o menor custo de inserção, $S_{\max}$ o maior e α um parâmetro que define o nível de aleatoriedade da escolha. De forma aleatória, escolhe-se dessa lista restrita de candidatos o próximo vértice k. Após a escolha, atualiza-se a solução a partir da inserção do novo vértice e retira-se k da lista de candidatos.

6.2.4 Algoritmo do Vizinho Mais Próximo

O Algoritmo do Vizinho Mais Próximo pode ser descrito da seguinte maneira. A partir de uma solução parcial contendo dois vértices escolhidos aleatoriamente, a cada passo o algoritmo insere na solução o vértice que for mais próximo do último vértice inserido. Uma variação desse algoritmo, conhecido como Algoritmo de Bellmore e Nemhauser, e adotada neste trabalho, consiste em analisar as distâncias dos vértices candidatos em relação aos vértices das extremidades e não somente em relação ao último vértice inserido.

O Algoritmo 12 mostra o pseudocódigo de uma dessa variante considerada do Algoritmo do Vizinho Mais Próximo.

Algoritmo 12 Heurística do Vizinho Mais Próximo

1: $s \leftarrow \{i - j\}, i, j \in T$ (Selecionados aleatoriamente);
2: $LC \leftarrow (V - \{i, j\})$;
3: **enquanto** $((\exists i \in W \, não \; coberto) \; || \; (\exists j \in T \notin s) \; || \; (\sum_{i \in s} p_i < PRIZE))$ **faça**
4: Defina o custo de inserção dos vértices de LC em relação ao vértices das extremidades de s;
5: Defina LRC;
6: $k \leftarrow$ vértice selecionado aleatoriamente de LRC;
7: $pos \leftarrow$ extremidade que k será inserido na rota;
8: $s \leftarrow s \cup \{k\}$;
9: $LC \leftarrow LC \setminus \{k\}$;
10: **fim enquanto**
11: Retorne s;

O algoritmo inicia com uma rota parcial com dois vértices pertencentes ao conjunto T, escolhidos aleatoriamente. Os vértices candidatos a entrarem na solução (vértices de T que ainda não foram inseridos na solução e os vértices de $V \setminus T$) são inseridos em uma lista de candidatos (LC). Construída a LC, enquanto todos os vértices de W não estiverem cobertos por algum vértice da solução, enquanto existir algum vértice de T fora da solução e o prêmio mínimo não tiver sido coletado, o algoritmo calcula a distância de todos os vértices pertencentes a LC aos vértices das extremidades. Os vértices k que possuírem a distância $d_{ik} \leq d_{\min} + \alpha \times (d_{\max} - d_{\min})$ são inseridos em LRC, sendo d_{ik} a menor distância entre k e uma das extremidades, $d_{\min}$ a menor distância entre k e as duas extremidades, $d_{\max}$ a maior distância e α um parâmetro que define o nível de aleatoriedade da escolha. De forma aleatória, escolhe-se dessa lista restrita de candidatos o próximo vértice k. Após a escolha, atualiza-se a solução a partir da inserção do novo vértice e retira-se k da lista de candidatos.

6.2.5 Heurística GENIUS

A heurística GENIUS foi desenvolvida para o PCV por [Gendreau *et al.*, 1992] e é composta por duas fases: uma de construção da rota, que consiste na inserção dos vértices (GENI - *Generalized Insertion*) e uma de melhoramento (US - *Unstringing and Stringing*), que consiste em remover os vértices e reinserí-los na tentativa de encontrar uma melhor posição do que aquela onde foi inserida na fase de inserção. Para cada fase do GENIUS, existem duas maneiras distintas de se inserir e remover os vértices respectivamente. O pseudocódigo do GENIUS é mostrado no Algoritmo 13.

Algoritmo 13 Heurística GENIUS

1: $s \leftarrow \{v_i - v_j - v_k\}, i, j, k \in T$ (Selecionados aleatoriamente);
2: $LC \leftarrow (V \setminus \{v_i, v_j, v_k\})$;
3: **enquanto** $((\exists v_i \in W$ *não coberto*$)$ || $(\exists v_j \in T \notin s)$ || $(\sum_{v_i \in s} p_i < PRIZE))$ **faça**
4: $v_t \leftarrow$ vértice selecionado aleatoriamente de LC;
5: $s \leftarrow$ GENI(s,v_t) (*Generalized Insertion*);
6: **fim enquanto**
7: $s \leftarrow$ US(s) (*Unstringing and Stringing*);
8: Retorne s;

Inicialmente, a heurística GENIUS constrói uma solução parcial contendo três vértices escolhidos aleatoriamente. Definidos esses três vértices, define-se a lista de candidatos (LC) a entrar na solução. Nessa lista entram todos os vértices pertencentes a V que ainda não fazem parte da rota inicial. Enquanto o critério de parada do algoritmo não for satisfeito (não cobertura dos vértices de W, vértices de T não pertencentes à solução e a não coleta do prêmio mínimo estabelecido), em cada iteração escolhe-se um vértice v_t para entrar na solução e invoca-se o procedimento **GENI** para definir o melhor posição de inserí-lo. Quando o critério de parada for satisfeito, a partir da solução construída pela fase anterior, a heurística tenta encontrar uma posição melhor para os vértices pertencentes à rota aplicando-se o procedimento **US**.

6.2.5.1 GENI - *Generalized Insertion*

O algoritmo 14 apresenta o pseudocódigo da fase GENI.

Algoritmo 14 FaseGENI(s,v_t)

```
1: Defina o conjunto NP_s dos vértices pertencentes a s;
2: Defina o conjunto NP_LC dos vértices pertencentes a LC;
3: para (v_i ∈ NP_LC(v_t)) faça
4:    para (v_j ∈ NP_LC(v_t), v_i ≠ v_j) faça
5:       para (sentidoInsercao ← {horário, anti-horário}) faça
6:          s' ← InsercaoTipoI(v_t,v_i,v_j,sentidoInsercao);
7:          se f(s') < f* então
8:             s* ← s';
9:          fim se
10:         s' ← InsercaoTipoII(v_t,v_i,v_j,sentidoInsercao);
11:         se f(s') < f* então
12:            s* ← s';
13:         fim se
14:      fim para
15:   fim para
16: fim para
17: Modifique s' inserindo vértice v_t na melhor posição entre todos os pares de vértices
    adjacentes;
18: se f(s') < f* então
19:    s* ← s';
20: fim se
21: s ← s*;
22: LC ← LC \ v_t;
23: Atualize NP_s;
24: Atualize NP_LC;
25: Retorne s;
```

A principal característica da fase GENI é que a inserção de um vértice v_t em uma rota não acontece necessariamente entre dois vértices consecutivos. Porém, após a inserção, esses dois vértices tornam-se adjacentes a v_t.

O algoritmo da fase GENI inicia com a construção do conjunto (NP_s). Esse conjunto armazena, para cada vértice $v_k \in s$, onde s representa a rota parcial, uma lista contendo os p vértices mais próximos de v_k que também já estão na solução, sendo p um parâmetro. É com base nessa lista que, ao inserir um vértice v_t na solução entre dois vértices v_i e v_j não-adjacentes, se consegue reorganizar a solução de maneira eficiente. Após a construção do conjunto NP_s, a próxima etapa consiste em contruir a lista NP_{LC} para todos os vértices que estão em LC. Da mesma forma que NP_s, NP_{LC} armazena, para cada vértice $v_t \in LC$, uma lista contendo os p vértices mais próximos de v_t que já fazem parte da solução. O objetivo de NP_{LC} é ajudar na inserção dos vértices que ainda não fazem parte da rota, de forma que eles sejam inseridos perto de seus vértices mais próximos. Para cada combinação dois a dois entre os vértices pertencentes à $NP_{LC}(v_t)$, tenta-se inserir v_t utilizando os dois

tipos de inserção (descritos posteriormente). A melhor inserção, isto é, a que trouxer o menor acréscimo no custo da solução é armazenada em s^*, onde s^* representa a solução que possui o menor custo com a inserção do vértice v_t. Após tentar inserir v_t pelos dois tipos de inserção, tenta-se inserir v_t entre dois vértices que sejam adjacentes na solução e se essa inserção trouxer o menor custo, atualiza-se s^*. Ao final da iteração, atualiza-se a solução s (isto é, $s \leftarrow s^*$), retira-se v_t de LC e de NP_{LC} e o adiciona a NP_s. Além disso, é preciso atualizar os p vértices mais próximos dos vértices de NP_s e NP_{LC} com v_t. Pode acontecer de v_t ser um vértice mais próximo de um vértice v_k que ainda não esteja na solução, mas como v_t também não fazia parte da solução, ele não pertencia à $NP_{LC}(v_k)$. O mesmo se aplica a NP_s. Por fim, retorna-se a nova solução parcial, incluindo agora o vértice v_t.

Com dito anteriormente, a fase GENI possui dois tipos de inserção. O Algoritmo 15 apresenta o pseudocódigo da inserção do tipo 1.

Inicialmente, o algoritmo verifica o caso trivial, isto é, se v_i e v_j são adjacentes. Se forem, insere-se v_t entre eles e retorna-se para o corpo principal do algoritmo GENI. Caso v_i e v_j não sejam adjacentes, define-se v_{i+1} e v_{j+1} como sendo os vértices subsequentes aos vértices v_i e v_j respectivamente, de acordo com sentido que está se inserindo. Por exemplo, se o sentido de inserção for o sentido horário, o vértice subsequente a v_i será o vértice posterior a v_i e se o sentido for o anti-horário, o vértice subsequente a v_i será o vértice anterior a v_i. Definidos v_{i+1} e v_{j+1}, o próximo passo é armazenar em $caminho_{ji}$ os vértices que estão no caminho entre v_j e v_i, de acordo com o sentido de inserção. Vale ressaltar que os vértices do caminho entre v_j e v_i não incluem v_j nem v_i. Conhecidos os vértices do caminho entre v_j e v_i, realiza-se a interseção entre esses vértices do caminho com aqueles que pertencem à lista dos p vértices mais próximos ao vértice v_{i+1}, vértice este pertencente ao conjunto dos que já estão na solução.

Algoritmo 15 GENI - InsercaoTipoI(v_t,v_i,v_j,*sentidoInsercao*)

1: $f' \leftarrow \infty$;
2: **se** (v_i,v_j e v_t forem adjacentes) **então**
3: $s_{curr} \leftarrow$ inserir v_t entre v_i e v_j;
4: retorne;
5: **fim se**
6: $v_{i+1} \leftarrow$ vértice subsequente a $v_i \in s_{curr}$, de acordo com o sentido de inserção;
7: $v_{j+1} \leftarrow$ vértice subsequente a $v_j \in s_{curr}$, de acordo com o sentido de inserção;
8: $caminho_{ji} \leftarrow$ vértices que estão no caminho de v_j a v_i;
9: $intersecao \leftarrow (v_k \in caminho_{ji}) \cap (v_k \in NP_s(v_{i+1}))$;
10: **se** ($intersecao = \emptyset$) **então**
11: retorne;
12: **fim se**
13: **para** ($v_k \in intersecao$) **faça**
14: $v_{k+1} \leftarrow$ vértice sequente a $v_k \in s_{curr}$, de acordo com o sentido de inserção;
15: Remover de s_{curr} as arestas que conectam (v_i,v_{i+1}),(v_j,v_{j+1}) e (v_k,v_{k+1});
16: Inserir em s_{curr} as arestas que conectam (v_i,v_t),(v_t,v_j),(v_{i+1},v_k) e (v_{j+1},v_{k+1});
17: **se** ($f_{curr} < f(s')$) **então**
18: $s' \leftarrow s_{curr}$;
19: **fim se**
20: **fim para**
21: Retorne s';

Se a interseção for vazia, não existe nenhum vértice em comum entre os dois conjuntos, então retorna-se ao corpo principal do algoritmo GENI. Caso contrário, isto é, existindo algum vértice comum aos dois conjuntos, para cada vértice v_k pertencente à interseção toma-se o vértice v_{k+1} como sendo aquele subsequente a v_k, de acordo com o sentido da inserção. Como todos os vértices necessários à inserção já estão definidos, o próximo passo consiste em remover algumas arestas entre esses vértices e inserir outras arestas. Se essa nova configuração da solução possuir um custo menor que o de s', então ela passa ser a nova s'. Note que as remoções e inserções de arestas para cada variação de v_k são realizadas em s_{curr}, enquanto que em s' armazena-se a melhor inserção realizada entre todos os diferentes v_k. Ao final do procedimento, retorna-se essa melhor solução (s') para o corpo principal do algoritmo GENI.

O Algoritmo 16 apresenta o pseudocódigo da inserção do tipo 2.

Analogamente ao Algoritmo 15, o procedimento de inserção do tipo 2 inicia verificando o caso trivial, isto é, se v_i e v_j são adjacentes. Se forem, insere-se v_t entre eles e retorna ao corpo principal do algoritmo GENI. Caso v_i e v_j não sejam adjacentes, define-se v_{i+1} e v_{j+1} como sendo os vértices subsequentes aos vértices v_i e v_j respectivamente, de acordo com sentido que está se inserindo.

Algoritmo 16 GENI - InsercaoTipoII(v_t,v_i,v_j,*sentidoInsercao*)

1: $f' \leftarrow \infty$;
2: **se** (v_i,v_j e v_t forem adjacentes) **então**
3: $s_{curr} \leftarrow$ inserir v_t entre v_i e v_j;
4: retorne;
5: **fim se**
6: $v_{i+1} \leftarrow$ vértice subsequente a $v_i \in s_{curr}$, de acordo com o sentido de inserção;
7: $v_{j+1} \leftarrow$ vértice subsequente a $v_j \in s_{curr}$, de acordo com o sentido de inserção;
8: $caminho_{ji} \leftarrow$ vértices que estão no caminho de v_j a v_i;
9: $intersecao_{ji} \leftarrow (v_k \in caminho_{ji}) \cap (v_k \in NP_s(v_{i+1}))$;
10: **se** ($intersecao_{ji} = \emptyset$) **então**
11: retorne;
12: **fim se**
13: $caminho_{ij} \leftarrow$ vértices que estão no caminho de v_i a v_j;
14: $intersecao_{ij} \leftarrow (v_l \in caminho_{ji}) \cap (v_l \in NP_s(v_{j+1}))$;
15: **se** ($intersecao_{ij} = \emptyset$) **então**
16: retorne;
17: **fim se**
18: **para** ($v_k \in intersecao_{ji}$) **faça**
19: $v_{k-1} \leftarrow$ vértice subsequente a $v_k \in s_{curr}$, de acordo com o sentido de inserção;
20: **para** ($v_l \in intersecao_{ij}$) **faça**
21: $v_{l-1} \leftarrow$ vértice subsequente a $v_l \in s_{curr}$, de acordo com o sentido de inserção;
22: Remova de s_{curr} as arestas que conectam (v_i,v_{i+1}),(v_j,v_{j+1}),(v_{l-1},v_l) e (v_{k-1},v_k);
23: Insira em s_{curr} as arestas que conectam (v_i,v_t),(v_t,v_j),(v_l,v_{j+1}),(v_{k-1},v_{l-1}) e (v_{i+1},v_k);
24: **se** ($f_{curr} < f(s')$) **então**
25: $s' \leftarrow s_{curr}$;
26: **fim se**
27: **fim para**
28: **fim para**
29: Retorne s';

Definidos v_{i+1} e v_{j+1}, o próximo passo é armazenar em $caminho_{ji}$ os vértices que estão no caminho entre v_j e v_i, de acordo com o sentido de inserção, à exceção desses extremos. Conhecidos os vértices desse caminho, realiza-se a interseção desses com aqueles que pertencem à lista dos p vértices mais próximos de v_{i+1}, vértice este pertencente a NP_s (conjunto dos vértices que já estão na solução). Se a interseção for vazia, não existe vértice em comum entre os dois conjuntos; então retorna-se ao corpo principal do algoritmo GENI. Se existir algum vértice comum aos dois conjuntos, o próximo passo é armazenar em $caminho_{ij}$ os vértices que estão no caminho entre v_i e v_j excluindo-se estes, de acordo com o sentido da inserção. Conhecidos os vértices do caminho entre v_i e v_j, realiza-se a interseção entre esses vértices do caminho com aqueles que pertencem à lista dos p vértices mais próximos do vértice v_{j+1}, vértice este também pertencente a NP_s.

Caso a interseção seja vazia, então retorna-se ao corpo principal do algoritmo GENI; caso contrário, para cada combinação de vértices v_k e v_l pertencentes a $intersecao_{ji}$ e $intersecao_{ij}$ respectivamente, toma-se v_{k-1} como sendo o vértice subsequente a v_k e v_{l-1} como sendo o vértice subsequente a v_l, ambos de acordo com o sentido da inserção. O próximo passo consiste em remover algumas arestas entre esses vértices e inserir outras arestas. Se essa nova solução tiver um custo menor que a de s', então ela passa ser a nova s'. Ao final do procedimento, retorna-se a melhor solução (s') para o corpo principal do algoritmo GENI.

6.2.5.2 US - *Unstringing and Stringing*

O algoritmo 17 apresenta o pseudocódigo da fase US da Heurística GENIUS.

Algoritmo 17 US(s)

```
1: f* ← f(s);
2: s' ← s;
3: para (v_i ∈ s) faça
4:    para (sentidoRemocao = {horário, anti-horário}) faça
5:       s' ← RemocaoTipoI(s',v_i, sentidoRemocao);
6:       se (f(s') < f*) então
7:          s* ← s';
8:       fim se
9:       s' ← s;
10:      s' ← RemocaoTipoII(s',v_i, sentidoRemocao);
11:      se (f(s') < f*) então
12:         s* ← s';
13:      fim se
14:   fim para
15:   se (f* < f(s)) então
16:      s ← s*;
17:   fim se
18: fim para
19: Retorne s;
```

A fase US consiste em, para cada vértice $v_k \in s$, retirá-lo da solução da melhor maneira possível utilizando-se um dos dois tipos de remoção nas duas orientações e reinserí-lo utilizando-se a fase de inserção da heurística GENIUS. Observe que a tentativa de reinserir v_k na solução ocorre para cada tipo de remoção e em cada sentido. Por exemplo, aplica-se a remoção do tipo 1 em v_k para o sentido horário e tenta-se reinserí-lo com a fase GENI, detalhada anteriormente. Após isso, aplica-se a remoção do tipo 2 no sentido horário e a inserção novamente. O mesmo procedimento é realizado para o sentido anti-horário.

Com dito anteriormente, a fase US também possui dois tipos de remoção. O Algoritmo 18 apresenta o pseudocódigo da remoção do tipo 1.

Algoritmo 18 US - RemocaoTipoI(s',v_i, *sentidoRemocao*)

1: $s_{base} \leftarrow s'$;
2: $v_{i-1} \leftarrow$ vértice subsequente a $v_i \in s_{base}$, de acordo com o sentido de remoção;
3: $v_{i+1} \leftarrow$ vértice subsequente a $v_i \in s_{base}$, de acordo com o sentido de remoção;
4: **para** ($v_j \in NP_s(v_{i+1})$) **faça**
5: $v_{j-1} \leftarrow$ vértice subsequente a $v_j \in s_{base}$, de acordo com o sentido de remoção;
6: $v_{j+1} \leftarrow$ vértice subsequente a $v_j \in s_{base}$, de acordo com o sentido de remoção;
7: *caminho* $\leftarrow$ vértices que estão no caminho de v_{i+1} a v_{j-1};
8: *intersecao* $\leftarrow$ ($v_k \in$ *caminho*) $\cap$ ($v_k \in NP_s(v_{i-1})$);
9: **se** (*intersecao* $= \emptyset$) **então**
10: retorne;
11: **fim se**
12: **para** ($v_k \in$ *intersecao*) **faça**
13: $s_curr \leftarrow s_{base}$;
14: $v_{k+1} \leftarrow$ vértice subsequente a $v_k \in s_{base}$, de acordo com o sentido de remoção;
15: Remover de s_{curr} as arestas que conectam (v_{i-1},v_i),(v_i,v_{i+1}),(v_k,v_{k+1}) e (v_j,v_{j+1});
16: Inserir em s_{curr} as arestas que conectam (v_{i-1},v_k),(v_{i+1},v_j) e (v_{k+1},v_{j+1});
17: $s_{curr} \leftarrow$ GENI(s_{curr},v_i);
18: **se** ($f_{curr} < f(s')$) **então**
19: $s' \leftarrow s_{curr}$;
20: **fim se**
21: **fim para**
22: **fim para**
23: Retorne s';

O algoritmo inicia com a atribuição dos vértices anterior e posterior ao vértice v_i, de acordo com o sentido de remoção em que está se trabalhando. Para cada vértice v_j pertencente à lista dos p vértices mais próximos a v_{i+1}, define-se também os vértices anterior e posterior ao vértice v_j, de acordo com o sentido de remoção. Definidos v_{j-1} e v_{j+1}, o conjunto *caminho* armazena todos os vértices que estiverem no caminho entre v_{i+1} e v_{j-1}, caminho esse também relativo ao sentido tratado, isto é, horário ou anti-horário. Já o conjunto *intersecao* armazena os vértices comuns a *caminho* e ao conjunto dos p vértices mais próximos de v_{i-1}. Se interseção não for vazia, para cada vértice v_k pertencente a este conjunto, toma-se o vértice posterior a v_k. Após definir todos os vértices necessários, é preciso remover e inserir algumas arestas de forma que v_i saia da solução sem deixar a rota desconexa. Por fim, aplica-se a fase de inserção (GENI) para reinserir v_i à solução e retorna-se ao corpo principal da fase US.

O Algoritmo 19 apresenta o pseudocódigo da remoção do tipo 2.

Algoritmo 19 US - RemocaoTipoII(s',v_i, *sentidoRemocao*)

$s_{base} \leftarrow s'$;
$v_{i-1} \leftarrow$ vértice subsequente a $v_i \in s_{base}$, de acordo com o sentido de remoção;
$v_{i-2} \leftarrow$ vértice subsequente a $v_{i-1} \in s_{base}$, de acordo com o sentido de remoção;
$v_{i+1} \leftarrow$ vértice subsequente a $v_i \in s_{base}$, de acordo com o sentido de remoção;
para ($v_j \in NP_s(v_{i+1})$) **faça**
  $v_{j-1} \leftarrow$ vértice subsequente a $v_j \in s_{base}$, de acordo com o sentido de remoção;
  $v_{j+1} \leftarrow$ vértice subsequente a $v_j \in s_{base}$, de acordo com o sentido de remoção;
  $caminho_{ji} \leftarrow$ vértices que estão no caminho de v_{j+1} a v_{i-2};
  $intersecao_{ji} \leftarrow (v_k \in caminho_{ji}) \cap (v_k \in NP_s(v_{i-1}))$;
  se ($intersecao_{ji} = \emptyset$) **então**
    retorne;
  fim se
  para ($v_k \in intersecao_{ji}$) **faça**
    $v_{k-1} \leftarrow$ vértice subsequente a $v_k \in s_{base}$, de acordo com o sentido de remoção;
    $v_{k+1} \leftarrow$ vértice subsequente a $v_k \in s_{base}$, de acordo com o sentido de remoção;
    $caminho_{jk} \leftarrow$ vértices que estão no caminho de v_j a v_{k-1};
    $intersecao_{jk} \leftarrow (v_l \in caminho_{jk}) \cap (v_l \in NP_s(v_{k+1}))$;
    se ($intersecao_{jk} = \emptyset$) **então**
      retorne;
    fim se
    para ($v_l \in intersecao_{jk}$) **faça**
      $s_curr \leftarrow s_{base}$;
      $v_{l+1} \leftarrow$ vértice subsequente a $v_l \in s_{base}$, de acordo com o sentido de remoção;
      Remova de s' as arestas que conectam (v_{i-1},v_i),(v_i,v_{i+1}),(v_k,v_{k+1}),(v_{j-1},v_j) e (v_l,v_{l+1});
      Insira em s' as arestas que conectam (v_{i-1},v_k),(v_{l+1},v_{j-1}),(v_{i+1},v_j) e (v_l,v_{k+1});
      $s_curr \leftarrow$ GENI(s_{curr},v_i);
      se ($f_{curr} < f(s')$) **então**
        $s' \leftarrow s_{curr}$;
      fim se
    fim para
  fim para
fim para
Retorne s';

Da mesma forma que o Algoritmo 18, a remoção do tipo 2 inicialmente define os vértices anterior e posterior ao vértice v_i, de acordo com o sentido de remoção em que se está trabalhando. Além de v_{i-1}, vértice anterior a v_i, é necessário também definir v_{i-2}, o vértice anterior a v_{i-1}, de acordo com o sentido tratado. Para cada vértice v_j pertencente à lista dos p vértices mais próximos a v_{i+1}, define-se também os vértice anteriores e sucessores ao vértice v_j, de acordo com o sentido de remoção. Definidos v_{j-1} e v_{j+1}, o conjunto $caminho_{ji}$ armazena todos os vértices que estiverem no caminho entre v_{j+1} e v_{i-2}, caminho esse também relativo ao sentido tratado, isto é, horário ou anti-horário. Já o

conjunto $intersecao_{ji}$ armazena os vértices comuns a $caminho_{ji}$ e ao conjunto dos p vértices mais próximos de v_{i-1}. Se interseção não for vazia, para cada vértice v_k pertencente a este conjunto, define-se também os vértices anterior e posterior ao vértice v_k, de acordo com o sentido de remoção. Definidos v_{k-1} e v_{k+1}, o conjunto $caminho_{jk}$ armazena todos os vértices que estiverem no caminho entre v_j e v_{k-1}, de acordo com o sentido tratado. Já o conjunto $intersecao_{jk}$ armazena os vértices comuns a $caminho_{jk}$ e ao conjunto dos p vértices mais próximos de v_{k+1}. Se interseção não for vazia, para cada vértice v_l pertencente a $intersecao_{jk}$, define-se v_{l+1}, posterior a v_l. Após definir todos os vértices necessários, é preciso remover e inserir algumas arestas de forma que v_i saia da solução sem deixar a rota desconexa. Por fim, aplica-se a fase de inserção (GENI) para reinserir v_i à solução, retornando-se ao corpo principal da fase US.

6.3 Estruturas de Vizinhança

Esta seção descreve as estruturas de vizinhança utilizadas no presente trabalho para explorar o espaço de soluções do problema. São utilizadas nove vizinhanças, nomeadas de N^1 a N^9, conforme a seguir. As cinco primeiras envolvem movimentos clássicos encontrados na literatura para o PCV e as quatro últimas são baseadas nas heurísticas GENIUS (Seção 6.2.5) e Inserção Mais Barata (Seção 6.2.3).

- N^1 - composta pelo movimento *shift*;
- N^2 - composta pelo movimento *swap*;
- N^3 - composta pelo movimento *Or-opt*;
- N^4 - composta pelo movimento *2-Opt*;
- N^5 - composta pelo movimento *3-Opt*;
- N^6 - composta pelo movimento *addGenius*;
- N^7 - composta pelo movimento *dropGenius*;
- N^8 - composta pelo movimento *addCheapestInsertion*;
- N^9 - composta pelo movimento *dropSimple*;

A primeira estrutura de vizinhança gera soluções vizinhas a partir do movimento *shift*, que consiste em remover um vértice e o reinserí-lo em uma outra posição, fazendo com que ele se ligue a dois novos vértices.

O movimento *swap* consiste em fazer a troca de posição entre dois vértices da solução.

O movimento *Or-Opt* transfere n vértices consecutivos na solução para uma outra posição na rota, sendo $2 \leq n \leq |V| - 2$. Esse movimento é uma extensão do movimento *shift*.

O quarto movimento, *2-Opt*, consiste em remover duas arestas não adjacentes da solução e inserir duas novas arestas de forma que continue existindo apenas um ciclo.

O movimento *3-Opt* possui o mesmo conceito do movimento *2-Opt*, consistindo na remoção de três arestas não adjacentes da solução e inserção de três novas arestas de forma que continue existindo apenas um ciclo. A diferença do movimento *3-Opt* é que este permite que se insira novas arestas de quatro maneiras diferentes, levando-se em conta que obrigatoriamente deve-se remover sempre três arestas. Se for levar em consideração a remoção de apenas duas arestas e manter uma fixa, existem mais três maneiras diferentes de inserção que podem ser obtidas através do movimento *2-Opt*. Observe então, que um movimento *3-Opt* é composto pelo movimento *2-Opt* somado com mais quatro maneiras distintas de se inserir novas arestas à solução.

O movimento *addGenius* consiste em inserir um vértice na solução por meio dos dois tipos de inserção (Algoritmos 15 e 16) da fase GENI (Seção 6.2.5.1) da heurística GENIUS.

Já o movimento *dropGenius* remove um vértice da solução utilizando os dois tipos de remoção (Algoritmos 18 e 19) propostos na fase US (Seção 6.2.5.2) da heurística GENIUS.

O movimento *cheapestInsertion* procura a melhor posição entre dois vértices adjacentes da solução para inserir um vértice. A posição que trouxer o menor custo adicional entre todos os pares de vértices adjacentes é escolhida.

O último movimento, *dropSimple*, consiste em remover um vértice da solução e inserir uma aresta ligando os dois vértices anteriormente adjacentes ao vértice removido.

Uma característica do PRRCP é que nem todos os vértices de $V \setminus T$ são sempre necessários à solução, alguns por motivos claros como, por exemplo, pelo fato de não cobrirem vértices de W e/ou o prêmio mínimo já estiver sido satisfeito.

Por outro lado, pode existir dúvida em determinar quais vértices de $V \setminus T$ farão parte da solução, pois tanto a utilização de um determinado vértice deste conjunto pode ser

melhor que a de outro deste conjunto, quanto a utilização de ambos na solução. Para solucionar tal problema, é preciso testar diferentes inserções e remoções de vértices que não fazem parte da solução corrente, como também remover e reinserir um mesmo vértice, pois pode ser que ele não se encontre em sua posição ideal.

Diante disso, foram criadas no presente trabalho oito combinações entre as quatro estruturas de vizinhança propostas, de forma a permitir inserir e remover novos vértices na tentativa de melhorar e, muitas vezes, viabilizar a solução. São elas:

- *dropGenius_addGenius*: retira um vértice utilizando a fase de remoção (US) e insere um novo vértice utilizando a fase de inserção (GENI), ambos da heurística GENIUS;
- *dropGenius_re_addGenius*: retira um vértice utilizando a fase de remoção (US) e o reinsere utilizando a fase de inserção (GENI), ambos da heurística GENIUS;
- *dropGenius_addCheapestInsertion*: retira um vértice utilizando a fase de remoção (US) da heurística GENIUS e insere um novo vértice com base na heurística da Inserção Mais Barata;
- *dropGenius_re_addCheapestInsertion*: retira um vértice utilizando a fase de remoção (US) da heurística GENIUS e o reinsere utilizando com base na Inserção Mais Barata;
- *dropSimple_addGenius*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e insere um novo vértice utilizando a fase de inserção (GENI) da heurística GENIUS;
- *dropSimple_re_addGenius*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e o reinsere utilizando a fase de inserção (GENI) da heurística GENIUS;
- *dropSimple_addCheapestInsertion*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e insere um novo vértice aplicando-se a heurística da Inserção Mais Barata;
- *dropSimple_re_addCheapestInsertion*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e o reinsere pela heurística da Inserção Mais Barata.

6.4 Função de Avaliação com Penalização

A avaliação de qualquer solução gerada por métodos exatos ou heurísticos utiliza a mesma função objetivo, como descrito nas formulações do PRRCP do Capítulo 4. Porém, para permitir que soluções inviáveis sejam aceitas nas heurísticas aqui propostas, adiciona-se à função objetivo, um termo que penaliza soluções inviáveis. Essa penalização é feita de modo que nas soluções ótimas e nas soluções viáveis o valor da função objetivo seja o mesmo, tanto para heurísticas como para os métodos exatos. Neste contexto, chama-se de função de avaliação penalizada a expressão dada na equação (6.3).

$$fp(s) = \sum_{i,j \in s} c_{ij} + \sum_{k \in R} \mu_k \times inv_k \tag{6.3}$$

em que:

- s : solução que está sendo avaliada;
- $fp(s)$: função de avaliação penalizada;
- R : conjunto de restrições descritas na Seção 3.1;
- c_{ij} : distância de um vértice i a um vértice j, com $i, j \in s$;
- μ_k : penalidade por desrespeitar a restrição $k \in R$;
- inv_k : número de vezes em que a restrição k é desrespeitada.

6.5 ILS aplicado ao PRRCP

Para refinar uma solução gerada pelos métodos descritos na Seção 6.2, foram desenvolvidos dois algoritmos híbridos. O primeiro, denominado ILS-MRD, combina a metaheurística *Iterated Local Search* (ILS) com o Método Randômico de Descida (MRD), sendo este último utilizado como um mecanismo de busca local para o ILS. Este algoritmo foi utilizado, com sucesso, em [Silva *et al.*, 2006] para tratar um problema de programação de jogos de competições esportivas. Já o segundo, ILS-VNRD, tem o VNRD (Seção 2.2.2) como mecanismo de busca local. O pseudocódigo do algoritmo ILS-MRD adaptado ao PRRCP é apresentado no Algoritmo 20.

Algoritmo 20 ILS-MRD()

1: $s_0 \leftarrow$ solução inicial gerada por um dos algoritmos descritos na Seção 6.2;
2: $s \leftarrow MRD(s_0, IterMRD)$;
3: $kp \leftarrow kp_0$;
4: **enquanto** $(kp < kp_{max})$ **faça**
5: **enquanto** $(iter - melhorIter < iter_{max})$ **faça**
6: $iter \leftarrow iter + 1$;
7: $s' \leftarrow perturbacao(s,kp)$;
8: $s'' \leftarrow MRD(s', IterMRD)$;
9: **se** $(f(s'') < f(s))$ **então**
10: $s \leftarrow s''$;
11: $melhorIter \leftarrow iter$;
12: $kp \leftarrow kp_0$;
13: **fim se**
14: **fim enquanto**
15: $kp \leftarrow kp + \delta$;
16: **fim enquanto**
17: Retorne s;

Esse algoritmo inicia-se gerando uma solução inicial aleatória, s_0, através de um dos métodos descritos na Seção 6.2. Em seguida, aplica-se um mecanismo de busca local, o Método Randômico de Descida (MRD) (Seção 2.2.1). Nesse método escolhe-se aleatoriamente um tipo de movimento (dentre os apresentados na Seção 6.3) e, a partir dele, é gerado um vizinho aleatório. Se esse vizinho for melhor que a solução corrente, ele passa a ser a nova solução corrente. Caso contrário, outro vizinho é gerado. O método MRD pára quando um número máximo de iterações sem melhora, no caso, *IterMRD*, for atingido. A cada iteração do método ILS-MRD, gera-se uma perturbação na solução corrente. Essa perturbação consiste em realizar k movimentos (dentre os apresentados na Seção 6.3), em uma estrutura de vizinhança escolhida aleatoriamente, sendo k um número arbitrário compreendido entre 1 e kp. A essa nova solução vizinha s', aplica-se o MRD, gerando-se uma solução da busca local, dada por s''. Se a solução s'' for melhor que a solução s corrente, então s'' é aceita como a nova solução corrente e reinicia-se o valor de kp. Essa repetição é executada até que $iter_{max}$ iterações sem melhora sejam realizadas. Quando isso ocorrer, incrementa-se o grau de perturbação, kp, em um fator δ, até que kp atinja seu valor máximo (kp_{max}). A variação do grau de perturbação é uma estratégia importante, pois permite a intensificação da busca e diversificação das soluções geradas. A busca é intensificada quando o grau de perturbação é baixo e a diversificação ocorre quando esse grau atinge um valor mais alto.

6.6 Algoritmo Evolutivo aplicado ao PRRCP

Além da heurística ILS, propõe-se um Algoritmo Evolutivo para resolver o PRRCP. Neste algoritmo, utiliza-se o conceito de classes para diferenciar os indivíduos. A classe A, representada pelo *pool* de soluções elite, contém os melhores indivíduos em termos de custo da solução, encontrados durante as gerações. Esses indivíduos são substituídos somente caso melhores indivíduos forem criados. A classe B é representada por uma parcela da população que também só é substituída caso indivíduos de melhor custo sejam gerados. Por fim, a classe C é representada pela outra parcela restante da população com indivíduos de custos mais elevados, substituídos em todas as gerações.

No Algortimo Evolutivo proposto, a população inicial é construída randomizando os métodos construtivos propostos na Seção 6.2. Como os métodos possuem idéias diferentes, diversificando-os na geração da população inicial, diversificam-se os indivíduos, promovendo, assim, uma miscigenação na população. Na fase de cooperação e adaptação, foram incorporados mecanismos de busca local e de intensificação para tentar melhorar os indivíduos gerados.

O Algoritmo 21 apresenta o pseudocódigo do Algoritmo Evolutivo proposto.

Neste algoritmo, enquanto os dois conjuntos (população e *pool* de soluções elite) não estiverem completos, são criados indivíduos utilizando-se os métodos para geração da solução inicial descritos na Seção 6.2. Cada indivíduo é então submetido a uma busca local rápida, realizada pelo MRD (Seção 2.2.1). Se esse indivíduo s for melhor que aquele que possuir o pior custo dentre todos do *pool* de soluções elite e possuir um percentual mínimo de diferença para todas as soluções do conjunto elite, então remove-se esse indivíduo e insere-se s no *pool*. Caso esse indivíduo não seja bom o bastante para entrar no conjunto elite, a mesma condição é realizada para verificar se ele pode fazer parte da população. Criada a população inicial, aleatoriamente escolhe-se uma solução da *populacao* para ser a solução base do procedimento *Reconexão por Caminhos* (explicado na Seção 6.7). A solução guia é selecionada do *pool* de soluções elite de acordo com o grau de diferença em relação à solução base. A solução que possuir o maior grau de diferença será escolhida como guia. A melhor solução encontrada durante o procedimento *Reconexão por Caminhos* é então submetida a uma busca local realizada pela metaheurística ILS-VNRD. Caso a solução s retornada pelo ILS-VNRD seja melhor que a solução de maior custo do conjunto de soluções elite e se s possuir uma porcentagem mínima de diferença para as demais soluções do conjunto elite, atualiza-se o *pool* inserindo s e removendo a pior solução. Ao

fim de cada geração, reconstrói-se uma parte da população (classe C), substituindo-se os piores indivíduos por novos indivíduos. Ao fim das gerações, retorna-se o melhor indivíduo (solução) do *pool*.

Algoritmo 21 AE-ILS-RC()

```
1: enquanto (!completo(populacao,pool)) faça
2:    s_0 ← solução inicial gerada por um dos algoritmos descritos na Seção 6.2;
3:    s ← MRD(s_0);
4:    pool ← s;
5:    se !(poolAtualizado) então
6:       populacao ← s;
7:    fim se
8: fim enquanto
9: enquanto (geracao < geracoes_max) faça
10:   s_base ← s ∈ populacao; (Escolhida aleatoriamente)
11:   s_guia ← s ∈ pool;
12:   Reconexão por Caminhos(s_base,s_guia);
13:   s_RC ← Reconexão por Caminhos(s_base,s_guia);
14:   s ← ILS-VNRD(s_RC);
15:   pool ← s;
16:   para (individuo < tamanhoClasseC) faça
17:      s_0 ← solução inicial gerada por um dos algoritmos descritos na Seção 6.2;
18:      s ← MRD(s_0);
19:      populacao ← s;
20:   fim para
21:   geracao ← geracao + 1;
22: fim enquanto
23: s ← s* ∈ pool;
24: Retorne s;
```

6.7 Reconexão por Caminhos (*Path Relinking*) aplicado ao PRRCP

O algoritmo Reconexão por Caminhos (RC) foi utilizado como fase de intensificação para o Algoritmo Evolutivo proposto na Seção 6.6. O objetivo do RC é tentar encontrar alguma solução melhor no trajeto entre a solução base e a guia. O Algoritmo 22 apresenta seu pseudocódigo.

Algoritmo 22 RC(s_{base})

```
1: maxDiff ← 0;
2: para (s ∈ pool) faça
3:    diff ← diferencaSimetrica(s,s_base);
4:    se (diff > maxDiff) então
5:       maxDiff ← diff;
6:       s_guia ← s;
7:    fim se
8: fim para
9: enquanto (diferencaSimetrica(s_base,s_guia) > 0) faça
10:   move ← moves[0];
11:   AplicarMovimento(move,s_base);
12:   s_cand ← s_base;
13:   s ← MRD(s_cand);
14:   s_pior ← indivíduo com o pior custo do pool de soluções elite;
15:   se (f(s) < f(s_pior)) então
16:      pool ← pool \ {s_pior};
17:      pool ← pool ∪ {s};
18:      poolAtualizado ← true;
19:   fim se
20: fim enquanto
```

O algoritmo inicia calculando a diferença simétrica de todas as soluções do *pool* de soluções elite em relação a s_{base}. A diferença simétrica pode ser entendida como a quantidade de passos que são necessários para sair de uma solução base e chegar à solução guia ou vice-versa. Sair de uma solução base e chegar a uma solução guia significa que a cada iteração, um atributo que exista na solução guia, mas que não pertença à solução base, é inserido na solução base com o objetivo de que ao final do procedimento a solução base coincida com a solução guia.

A fórmula para o cálculo da diferença simétrica entre duas soluções é dada por:

$$DS\left(s_{guia}, s_{base}\right) = \frac{a+v}{|A|+|V|} \tag{6.4}$$

em que:

- a representa o número de arestas semelhantes das soluções;
- v representa o número de vértices semelhantes das soluções;
- $A = \left(A^{guia} \cup A^{base}\right) \setminus \left(A^{guia} \cap A^{base}\right)$ representa o número de arestas distintos entre as soluções;

- $V = \left(V^{guia} \cup V^{base}\right) \setminus \left(V^{guia} \cap V^{base}\right)$ representa o número de vértices distintos entre as soluções.

Definida s_{guia}, enquanto s_{base} possuir alguma diferença para s_{guia}, o algoritmo aplica um movimento que consiste em inserir em s_{base} um atributo de s_{guia}. O movimento escolhido é o melhor dentre os possíveis movimentos que podem ser aplicados, isto é, o que trouxer o maior benefício para a solução. A essa solução intermediária, também chamada de candidata (s_{cand}), aplica-se uma busca local por meio do MRD. Se essa solução s for melhor que a pior solução do *pool* e atender ao critério de diversificação de soluções, então remove-se a pior solução do conjunto de soluções elite e insere-se s neste. Ao final da busca local, retorna-se à solução intermediária e prossegue-se com o algoritmo.

Para exemplificar o funcionamento do algoritmo Reconexão de Caminhos aplicado ao PRRCP, considere as Figuras 6.4 e 6.5 que mostram, respectivamente, uma solução base e uma solução guia para um problema envolvendo 29 cidades.

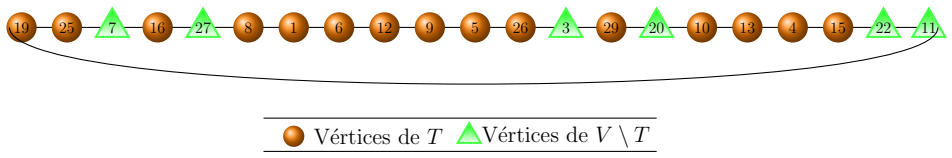

Figura 6.4: Solução base.

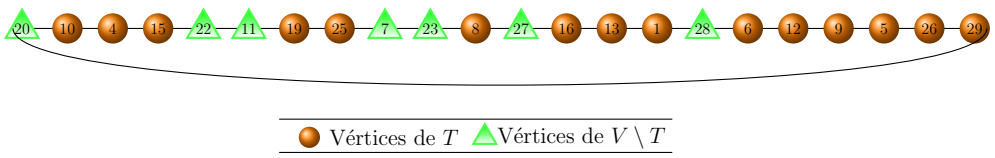

Figura 6.5: Solução guia.

Neste exemplo, deseja-se sair da solução base e com uma sequência de movimentos aplicados a essa solução, chegar à solução guia.

Como dito anteriormente, no PRRCP, uma solução não precisa conter obrigatoriamente todos os vértices de $V \setminus T$. Então, nesse problema, é possível ter diferentes soluções contendo, além dos vértices obrigatórios, diferentes vértices opcionais.

Para sair da solução base e alcançar a solução guia, é preciso adicionar e remover vértices das soluções, além de inserir e remover novas arestas nas soluções.

Para isso, distinguem-se dois tipos de movimentos: o primeiro consiste em remover todos os vértices que estão na solução base e não pertencem à solução guia. Com este movimento, além dos vértices, consegue-se remover arestas que não fazem parte da solução guia, pois ao remover um vértice, o arco que conectava este a um outro vértice passa a não existir mais na solução. O segundo movimento define todas as arestas existentes na solução guia que precisam ser inseridas na solução base. Com este movimento, vértices que não estão presentes na solução base são incluídos nela, pois inserir um arco implica em ligar dois vértices distintos.

Para exemplificar o cálculo da diferença simétrica entre duas soluções, observe na Figura 6.6 as diferenças existentes entre as soluções base e guia.

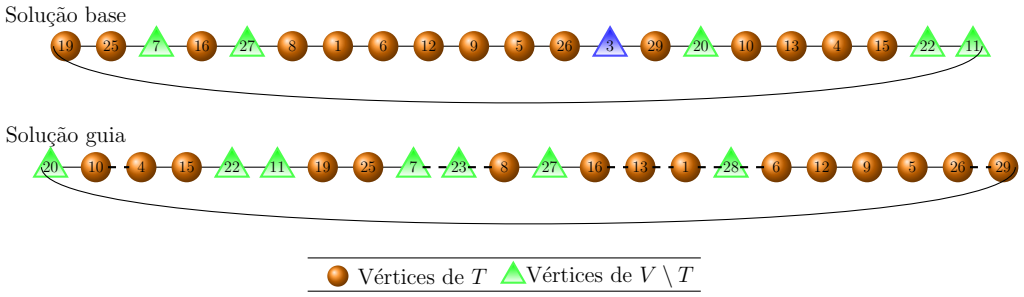

Figura 6.6: Diferença simétrica entre soluções.

Nessa figura, as linhas segmentadas na solução guia representam as arestas que devem ser inseridas na solução base e o vértice em azul representa o vértice da solução base que deve ser removido. Para definir a diferença simétrica entre as soluções guia e base, é preciso fazer alguns cáculos relacionados à quantidade de vértices e arestas que estão presentes nas duas soluções bem como a quantidade de vértices e arestas distintos entre as soluções. Assim, o cálculo da diferença simétrica entre as soluções base e guia é feito pela aplicação da expressão (6.4), que resulta, no exemplo considerado, em:

$$DS\left(s_{guia}, s_{base}\right) = \frac{14+19}{15+3} = \frac{33}{18} \cong 1,84 \tag{6.5}$$

Note que a solução guia contém mais vértices que a solução base, mas com a aplicação dos movimentos, o número de vértices em ambas as soluções se iguala. Por exemplo, a solução guia contém o vértice 23 que ainda não faz parte da solução base, mas com o movimento de inserção de arestas, as arestas que ligam os vértices 7 ao 23 e o vértice 23 ao 8 respectivamente, são inseridas na solução base. Ao se inserir arestas em que um

dos vértices não está presente na solução, insere-se tal vértice na solução. No exemplo, o vértice 23 é inserido na solução base.

Para exemplificar os dois tipos de movimentos, suponha que na primeira iteração do algoritmo o movimento que traz o maior benefício para a solução base dentre os nove possíveis seja o de retirar o vértice 3 da solução base. O movimento de retirar vértice apenas remove tal vértice e liga seus vértices adjacentes. A Figura 6.7 exemplifica o movimento de remoção do vértice 3 da solução base.

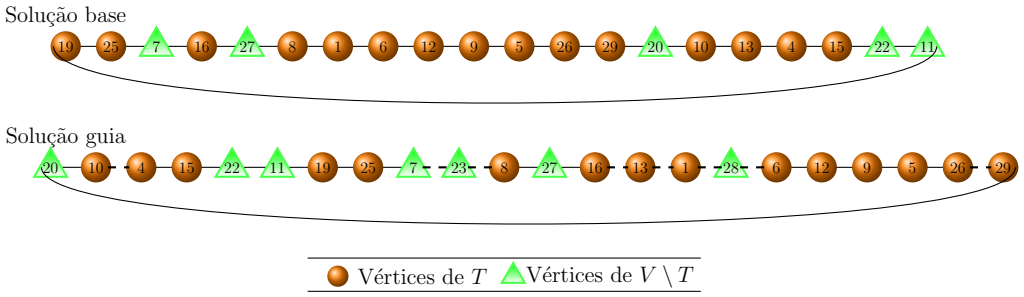

Figura 6.7: Remoção do vértice 3 da solução base.

Em todas as iterações é preciso redefinir todos o movimentos que precisam ser aplicados à solução base, pois ao aplicar um certo movimento, a solução pode sofrer alterações de modo que implicitamente algum outro movimento também seja aplicado, reduzindo assim, a quantidade de movimentos para se alcançar a solução guia. É o que acontece com a remoção do vértice 3 apresentado na Figura 6.7. Observe nessa figura que com a aplicação do movimento de remoção do vértice 3, os vértices 26 e 29 se ligaram, sendo que esta ligação era justamente um outro movimento possível.

Suponha agora, que na segunda iteração do algoritmo, o melhor movimento a ser aplicado consiste na inclusão do arco que liga o vértice 7 ao 23 na solução base. Note que para inserir um arco na solução é preciso saber a posição correta para inserção. Como o arco liga o vértice 7 ao 23, a interpretação é feita pensando que, por mais que o grafo não seja direcionado, a origem do arco é o vértice 7 e o destino é o vértice 23. Então, o vértice 23 tem que ser colocado à direita do vértice 7. Caso o arco ligasse o vértice 23 ao 7, o vértice 23 seria posicionado à esquerda do vértice 7. A Figura 6.8 exemplifica a inclusão do arco (7, 23) à solução base.

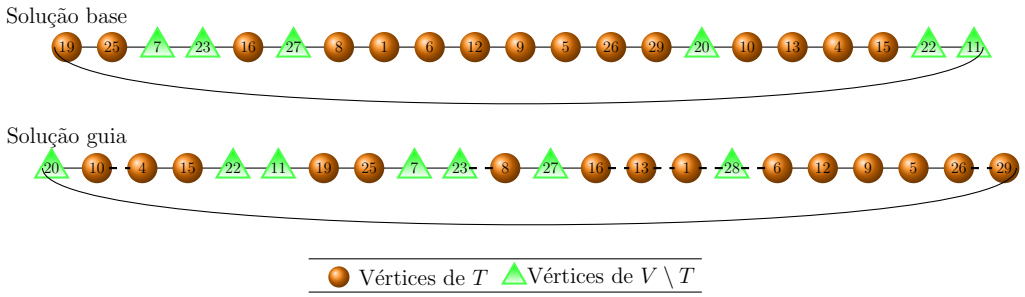

Figura 6.8: Inserção do arco (7,23) à solução base.

Por fim, para ilustrar um terceiro caso, suponha agora que o melhor movimento consiste em inserir à solução base o arco que liga o vértice 10 ao vértice 4. Da mesma forma que a inclusão do arco (7, 23), o vértice 4 é inserido à direita do vértice 10. Nesse caso, no entanto, não basta apenas tornar o vértice 4 adjacente ao vértice 10, pois de acordo com a Figura 6.8, o fragmento de rota que vai do vértice 4 ao vértice 23 já está posicionado corretamente de acordo com a solução guia. Sendo assim, é preciso trazer junto ao vértice 4, todos os vértices desse fragmento. A Figura 6.9 mostra como fica a solução base após a inclusão do arco (10, 4).

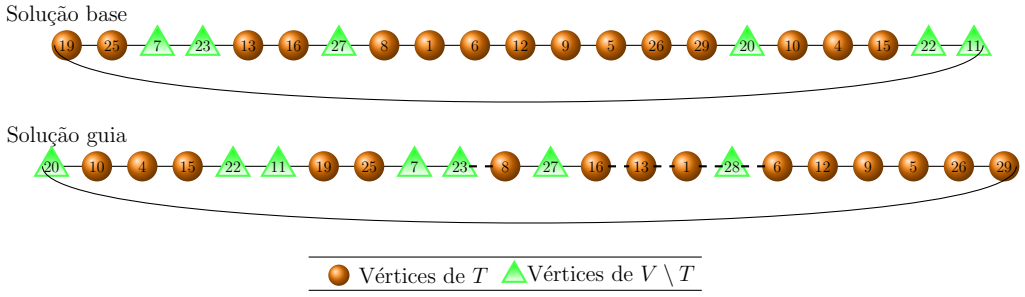

Figura 6.9: Inserção do arco (10,4) à solução base.

Ao final do algoritmo, a solução base é idêntica à solução guia, conforme desejado. A Figura 6.10 ilustra as duas soluções após a aplicação de todos os movimentos possíveis à solução base, isto é, até que a diferença simétrica entre as soluções seja zero.

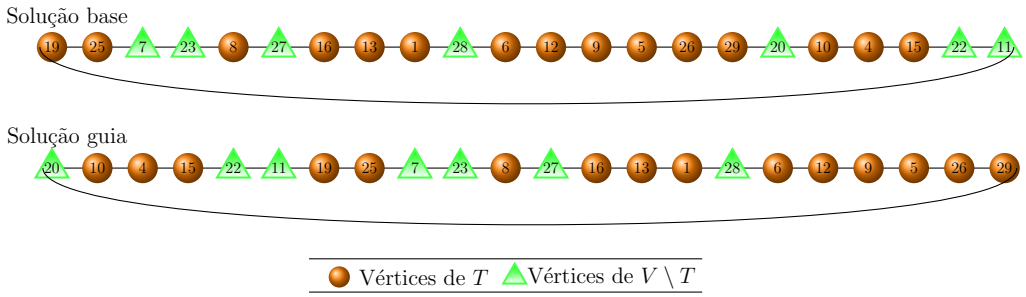

Figura 6.10: Resultado final do Algoritmo Reconexão por Caminhos.

Capítulo 7

Geração dos problemas-teste para o PRRCP

Como não foram encontrados problemas-teste na literatura para o PRRCP, descreve-se neste capítulo uma adaptação dos problemas-teste relativos ao Problema do Caixeiro Viajante para uso no PRRCP. A Seção 7.1 mostra como os diferentes tipos de vértices (T, $V \setminus T$ e W) foram definidos, enquanto a Seção 7.2 apresenta como os prêmios de cada vértice pertencente a V foram determinados. Na Seção 7.3 são apresentadas as características do conjunto de problemas-teste gerado.

7.1 Determinação dos vértices de T, $V \setminus T$ e W

Esta seção descreve como os três tipos de vértices que o PRRCP possui foram determinados, a partir de um problema-teste para o PCV.

O PRRCP, diferentemente do PCV, possui três tipos de vértices com características bem diferentes. O primeiro tipo, os vértices pertencentes ao conjunto T são os vértices obrigatórios. Esses vértices devem estar presentes em qualquer solução viável para o problema, supõe-se que eles podem cobrir um ou mais vértices do conjunto W, além de possuírem um prêmio associado. O segundo tipo de vértice, os pertencentes ao conjunto $V \setminus T$, são chamados de vértices opcionais, pois ao contrário dos vértices de T, não precisam necessariamente fazer parte de todas as soluções. Esses vértices também podem cobrir ou não um ou mais vértices de W e também possuem um prêmio associado. O último tipo é formado pelos vértices pertencentes ao conjunto W, os quais precisam ser cobertos pelos vértices de T e $V \setminus T$. Ao contrário dos outros dois tipos de vértices, não possuem prêmio associado.

No Problema do Caixeiro Viajante (PCV), existe apenas um tipo de vértice, pois

todos eles devem estar presentes em qualquer solução viável do problema. Desta forma, para adaptar um problema-teste do PCV para o PRRCP é preciso dividir os vértices do PCV em três grupos disjuntos de vértices.

Para gerar os problemas-teste do presente trabalho, adaptou-se um conjunto de problemas-teste disponíveis no repositório TSPLIB [Reinelt, 1991], a biblioteca de problemas-teste mais conhecida para o PCV.

Basicamente, a adaptação consiste em, a partir da matriz de distâncias dos vértices, das porcentagens de vértices que se deseja ter em cada um dos três conjuntos, encontrar uma distância de cobertura de forma que se consiga ter a porcentagem desejada de vértices de W com pelo menos algum vértice de V cobrindo esses vértices. Além disso, tomou-se outros cuidados, como por exemplo, de não permitir que apenas os vértices de T consigam cubrir os vértices de W e nem permitir que o prêmio mínimo a ser coletado possa ser satisfeito apenas com os vértices de T, pois caso esses casos aconteçam, descaracteriza-se o PRRCP, que se reduziria a um PCV, mas com apenas os vértices obrigatórios.

Para exemplificar a adaptação de um problema-teste do TSPLIB para o PRRCP, suponha um problema para o PCV contendo 10 vértices. A Tabela 7.1 apresenta as coordenadas cartesianas dos vértices.

Coordenada	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
x	56,5	39,0	34,5	43,0	70,5	65,0	50,0	52,5	58,0	65,0
y	57,5	48,5	75,0	92,5	65,5	82,0	65,0	100,0	117,5	113,0

Tabela 7.1: Coordenadas dos vértices.

Neste exemplo, deseja-se obter os conjuntos T, $V \setminus T$ e W com 3, 3, e 4 vértices respectivamente. A definição do tamanho de cada conjunto de vértices foi feita arbitrariamente, tentando-se ora ter muitos vértices obrigatórios, ora ter muitos vértices opcionais.

O primeiro passo consiste em determinar a matriz de distâncias desse conjunto de vértices. Para definir a distância entre dois vértices utilizando-se as coordenadas cartesianas, utiliza-se a métrica euclidiana.

A Tabela 7.2 apresenta a matriz de distâncias para os vértices do exemplo dado na Tabela 7.1.

Definida a matriz de distâncias entre os vértices, o próximo passo consiste em definir uma distância de cobertura D de tal forma que se consiga obter vértices suficientes para compor os três conjuntos, de forma que todos os vértices pertencentes ao conjunto W

Vértices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**0**	20	28	38	16	26	10	43	60	56
2	20	**0**	27	44	36	42	20	53	72	70
3	28	27	**0**	19	37	31	18	31	49	49
4	38	44	19	**0**	39	24	28	12	29	30
5	16	36	37	39	**0**	17	21	39	53	48
6	26	42	31	24	17	**0**	23	22	36	31
7	10	20	18	28	21	23	**0**	35	53	50
8	43	53	31	12	39	22	35	**0**	18	18
9	60	72	49	29	53	36	53	18	**0**	8
10	56	70	49	30	48	31	50	18	8	**0**

Tabela 7.2: Matriz de distâncias dos vértices.

estejam a uma distância menor ou igual a D de pelo menos um vértice de V.

Para isso, testa-se em ordem crescente de valor, as diferentes distâncias pertencentes à matriz de distâncias até que se encontre uma que seja possível determinar os vértices dos três conjuntos. Suponha $D = 18$. A próxima etapa consiste em definir quais pares de vértices estão a uma distância menor ou igual a 18. A Tabela 7.3 apresenta a lista de pares de vértices que possuem a distância inferior ou igual a 18.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
(1,5)	(1,7)	(3,7)	(4,8)	(5,6)	(8,9)	(8,10)	(9,10)

Tabela 7.3: Pares de vértices com $D \leq 18$.

Com base na Tabela 7.3, percebe-se que o vértice 2 não está a uma distância menor igual a 18 de nenhum outro vértice. Com isso, pode-se concluir que o vértice 2 não pode ser um vértice pertencente ao conjunto W, pois, com $D = 18$, não existe nenhum vértice que poderia cobrir o vértice 2.

Definidos os pares de vértices com distância menor ou igual a 18, define-se, para cada vértice pertencente à lista da Tabela 7.3, quais outros vértices, também presentes nessa lista, estão a uma distância menor ou igual a 18. Por exemplo, pela tabela existem duas arestas com a distância até 18 que ligam o vértice 1 aos vértices 5 e 7. Essa análise serve para definir quais vértices possuem mais vértices perto de si. A Tabela 7.4 mostra a lista dos vértices mais próximos de cada vértice pertencente à lista da Tabela 7.3.

1	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
5	7	8	1	5	3	4	8	8
7			6		1	9	10	9
						10		

Tabela 7.4: Lista dos vértices mais próximos entre si.

A partir da lista da Tabela 7.4, inicia-se a definição de qual conjunto cada vértice irá pertencer. Inicialmente, definem-se os vértices que irão compor o conjunto T. Esses vértices serão os que irão possuir o menor número de vértices próximos a eles. A razão pela qual os vértices de T são os que possuem o menor número de vértices próximos se dá justamente para dificultar um pouco a resolução do problema, pois se esses vértices forem os que possuírem mais vértices próximos, provavelmente eles por si só poderão cobrir todos os vértices de W, então o problema consistirá apenas em coletar o prêmio mínimo, pois a cobertura dos vértices de W estará garantida utilizando-se apenas os vértices de T.

Para cada vértice alocado ao conjunto T, atualizam-se todos os outros vértices que o tinham como próximo. Essa atualização se faz necessária pois um vértice que inicialmente possui um número maior de vértices próximos também pode ser um candidato a fazer parte de T, devido às atualizações realizadas.

Por exemplo, de acordo com a Tabela 7.4, os vértices 3, 4 e 6 são aqueles que possuem o menor número de vértices próximos a eles. Então, de acordo com a idéia proposta, o primeiro vértice a ser alocado a T será o 3. A seguir, atualiza-se a lista de todos os outros vértices que possuem o vértice 3 como próximo a eles. Neste exemplo, a lista do vértice 7 é atualizada. Com isso, o número de vértices próximos ao 7 diminui, fazendo com que este seja mais um candidato a fazer parte de T. O mesmo procedimento é feito para os vértices 4 e 6. Vale ressaltar que, quando se atualiza a lista dos vértices, reinicia-se a pesquisa do início da lista, pois como dito anteriormente, algum vértice que não era candidato a fazer parte de T, com a atualização pode passar a ser.

A Tabela 7.5 mostra a lista apresentada na Tabela 7.4 com as atualizações e a Figura 7.1 apresenta os vértices alocados a T.

1	**5**	**7**	**8**	**9**	**10**
5	1	1	9	8	8
7			10	10	9

Tabela 7.5: Lista dos vértices mais próximos após a formação do conjunto T.

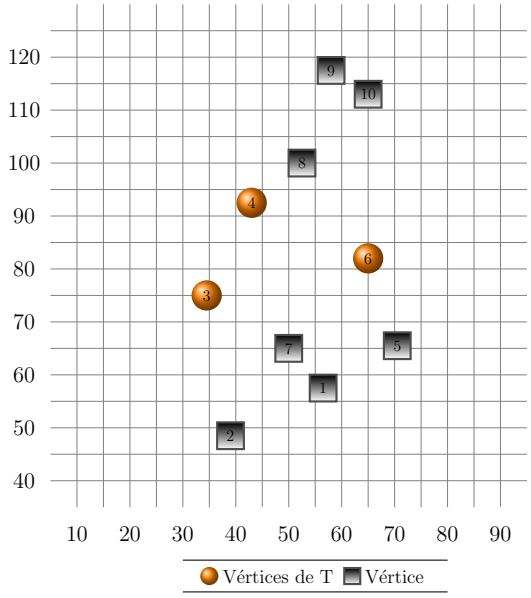

Figura 7.1: Representação dos vértices no plano cartesiano após definição dos vértices de T.

O conjunto W é o próximo a ser definido. Ao contrário da definição do conjunto T, os primeiros vértices a compor o conjunto W são os que possuírem o maior número de vértices próximos a ele. A justificativa para se escolher os vértices de W dessa forma é para aumentar a possibilidade de cobertura do vértice e evitar que seja direta a definição de qual vértice irá cobrí-lo.

Exemplificando, na Tabela 7.5 há quatro vértices com mesma quantidade de vértices próximos a eles. De acordo com a idéia proposta, o vértice 1 será o primeiro a ser alocado a W. Após isso, atualizam-se todos os outros vértices que possuem o vértice 1 como próximo a eles. Nesse caso, os vértices 5 e 7 são atualizados. Seguindo o procedimento, aloca-se o vértice 8 a W e atualizam-se os vértices 9 e 10. Esse procedimento de alocação prossegue até que se tenha o conjunto W formado.

A Tabela 7.6 mostra a lista apresentada na Figura 7.5 com as atualizações e a Figura 7.2 apresenta os vértices do conjunto W.

5	**7**	**10**
–	–	–

Tabela 7.6: Lista dos vértices mais próximos após a formação do conjunto W.

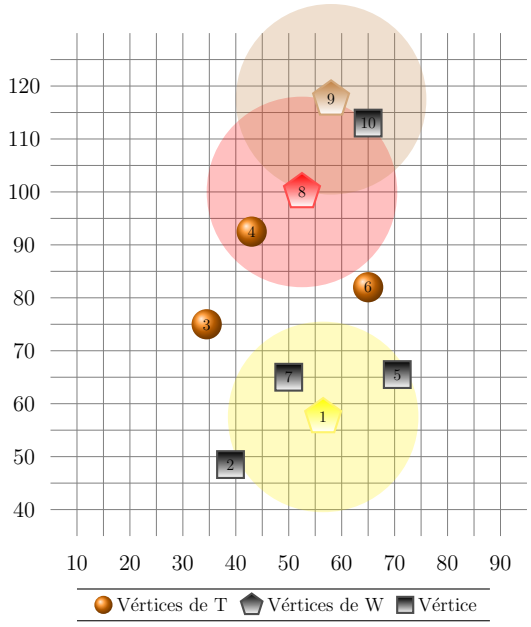

Figura 7.2: Representação dos vértices no plano cartesiano após alocação dos vértices de W.

De acordo com a Figura 7.2 e com o raio de cobertura em que os vértices de W podem ser cobertos, já pode-se definir quais vértices de V cobrirão quais vértices de W.

Por fim, são definidos os vértices do conjunto $V \setminus T$ como sendo os vértices restantes que não pertencem nem a T nem a W. No exemplo dado, o conjunto $V \setminus T$ é formado pelos vértices 2, 5, 7 e 10.

A Figura 7.3 descreve o problema-teste quase adaptado por completo para o PRRCP, isto é, com todos os vértices definidos a qual conjunto pertencem. A última etapa consiste em definir os prêmios associados a cada vértice de V, o que é explicado na Seção 7.2.

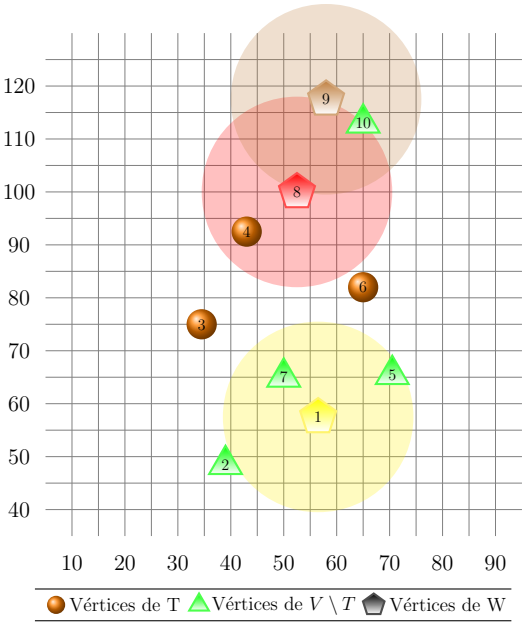

Figura 7.3: Representação dos vértices no plano cartesiano após definição dos vértices de $V \setminus T$.

Com base na Figura 7.3, conclui-se que o problema-teste criado (pela adaptação de um problema-teste do PCV), é válido para o PRRCP no quesito de cobertura de vértices, pois todos os vértices de W possuem pelo menos um vértice de V que os cobrem.

7.2 Determinação dos prêmios dos vértices de T e $V \setminus T$

Para definir os prêmios associados aos vértices dos conjuntos T e $V \setminus T$, utiliza-se a seguinte fórmula:

$$p_i = \left\lceil \frac{\sum\limits_{j \in (V \setminus \{i\})} (c_{ij})^2}{\sum\limits_{j \in (V \setminus \{i\})} (c_{ij})} \times \frac{\sum\limits_{j \in (V \setminus \{i\})} (c_{ij})}{|V| - 1} \times \frac{1}{\omega} \right\rceil \tag{7.1}$$

em que:

- c_{ij} é a distância entre o vértice i ao vértice $j \in V$, obtida na matriz de distâncias;
- V é o conjunto de vértices que podem fazer parte da solução, isto é, $V = T \cup V \setminus T$;
- ω é um fator que diminui a ordem de grandeza do valor do prêmio.

O objetivo de se realizar esse cálculo é beneficiar, de alguma forma, os vértices que estejam mais distantes, na média, dos demais vértices que esses possam se ligar, formando uma aresta. Dessa forma, supõe-se que os vértices mais distantes serão interessantes à solução, não por cobrirem algum outro vértice, mas sim por ajudar na coleta do prêmio mínimo pré-estabelecido.

Para exemplificar a equação, suponha o cálculo do prêmio referente ao vértice 2. O cálculo é realizado da seguinte forma:

$$p_2 = \left\lceil \frac{(27^2 + 44^2 + 36^2 + 42^2 + 20^2 + 70^2)}{(27 + 44 + 36 + 42 + 20 + 70)} \times \frac{(27 + 44 + 36 + 42 + 20 + 70)}{6} \times \frac{1}{\omega} \right\rceil \quad (7.2)$$

$$p_2 = \left\lceil \frac{(11025)}{(239)} \times \frac{(239)}{6} \times 0.01 \right\rceil = \lceil 46.13 \times 39.8 \times 0.01 \rceil = \lceil 18.36 \rceil = 19 \quad (7.3)$$

Observe nesse cálculo, que ω foi definido como 100 pelo fato de o numerador da segunda parcela da equação ser da ordem de grandeza das centenas.

A Figura 7.4 apresenta o problema-teste adaptado para o PRRCP. Os valores acima dos vértices obrigatórios e opcionais indicam o prêmio associado a cada vértice.

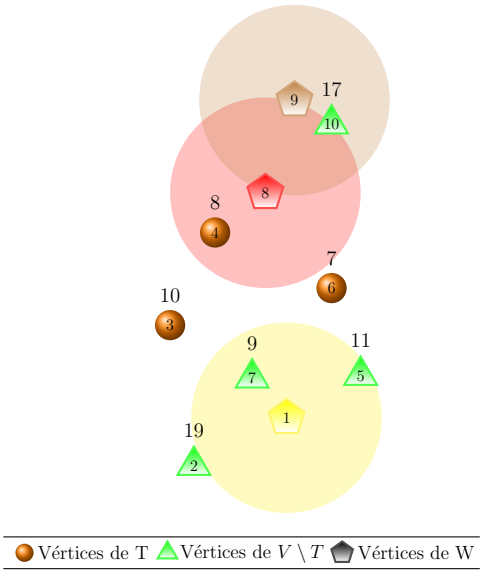

Figura 7.4: Problema-teste válido criado para o PRRCP.

Observe, na Figura 7.4, o funcionamento da fórmula de definição dos prêmios de cada vértice. O vértice 2, apesar de não cobrir nenhum vértice de W, é um vértice promissor

a fazer parte da solução, pois justamente por estar mais distante dos demais vértices, seu prêmio é mais elevado em relação aos demais vértices.

Foram criados problemas-teste com coleta mínima de 25%, 50% e 75% do prêmio total dos vértices de V. Com isso, em grande parte deles, alguns vértices de $V \setminus T$ devem estar obrigatoriamente presentes na solução, o que dificulta a resolução do problema, pois o número de combinações aumenta exponencialmente.

7.3 Problemas-teste

A fim de facilitar o entendimento do trabalho, os problemas-teste foram divididos em dois grupos:

Grupo 1: Formado por 111 problemas-teste com $48 \leq |V| \leq 200$.

Grupo 2: Formado por 33 problemas-teste com $201 \leq |V| \leq 400$.

7.3.1 Nomenclatura dos problemas-teste

Para nomear os problemas-teste, foi utilizado o próprio nome utilizado no TSPLIB juntamente com a informação relativa à porcentagem de cada conjunto de vértices ($V \setminus T$, T e W) e à porcentagem mínima de prêmio que a ser coletado. Por exemplo, o problema-teste `att48_VT9_T10_W29_25` se refere a um cujo nome no TSPLIB é `att48`. Nele, dos 48 vértices que o compõe, 9 pertencem ao conjunto dos vértices opcionais ($V \setminus T$), 10 fazem parte do conjunto de vértices obrigatórios (T) e 29 representam o conjunto de vértices que precisam ser cobertos. Além disso, pelo menos 25% dos prêmios associados aos vértices (T) e ($V \setminus T$) devem ser coletados.

A descrição detalhada de todos os problemas-teste encontram-se disponíveis no endereço http://www.ic.uff.br/∼satoru/conteudo/artigos/Resultados%20Problemas-teste%20PRRCP.pdf.

Capítulo 8

Resultados Computacionais

Este capítulo descreve os resultados computacionais obtidos com a formulação matemática, proposta no Capítulo 4, bem como os resultados obtidos com os algoritmos heurísticos apresentados no Capítulo 6. Primeiramente são apresentados os resultados obtidos aplicando-se as regras de redução propostas no Capítulo 5. Em seguida, são mostrados os resultados da aplicação da formulação de programação matemática. Finalmente, são apresentados e comparados os resultados da aplicação dos diferentes algoritmos heurísticos desenvolvidos, incluindo-se uma análise de distribuição de probabilidade empírica de se alcançar um dado valor alvo (isto é, encontrar uma solução com um custo no mínimo tão bom quanto o valor alvo) em função do tempo.

Os testes computacionais foram realizados em um computador Intel Core 2 Duo, com 2,2 GHz e 2,5 GB de memória principal, rodando o sistema operacional Windows Vista. Os algoritmos foram implementados na linguagem de programação C++, utilizando-se o ambiente de programação Microsoft Visual Studio 2008.

8.1 Resultados da aplicação das Regras de Redução

Os testes de redução foram aplicados em todos os problemas-teste. A porcentagem do número de vértices reduzidos foi, em média, de 23,2% em relação ao número original de vértices de cada problema.

A Tabela 8.1 apresenta a porcentagem de redução dos problemas-teste de cada grupo em que pelo menos um vértice foi reduzido. Nessa tabela, a primeira coluna se refere ao grupo de problemas-teste, a segunda coluna mostra o número de problemas-teste de cada grupo e a terceira coluna apresenta a taxa de sucesso por cada regra de redução, isto é, o

percentual de problemas-teste em que pelo menos um vértice foi reduzido pela aplicação de uma dada regra de redução.

Tabela 8.1: Taxa de sucesso das regras de redução

Porcentagem de problemas-teste reduzidas								
Grupo	Número de problemas-teste	Sucesso (%)						
-	-	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
`Grupo 1`	111	0	0,9	0	0	0	100	0
`Grupo 2`	33	0	0	0	0	0	100	0

A Tabela 8.2 apresenta a porcentagem média de redução de acordo com o tipo de vértice ($V \setminus T$, T e W) referente a cada problema-teste por grupo. Na primeira coluna mostra-se o nome da regra de redução; na segunda e terceira, a porcentagem média de redução de cada tipo de vértice dos problemas-teste dos Grupos 1 e 2, respectivamente.

Tabela 8.2: Porcentagem média redução por tipo de vértice e regra de redução

Porcentagem dos vértices reduzidos						
-	**Grupo 1**			**Grupo 2**		
Regra de Redução	$(V \setminus T)$	(T)	(W)	$(V \setminus T)$	(T)	(W)
`R1`	0	0	0	0	0	0
`R2`	0,0001	0	0	0	0	0
`R3`	0	0	0	0	0	0
`R4`	0	0	0	0	0	0
`R5`	0	0	0	0	0	0
`R6`	0	0	25,2	0	0	22,7
`R7`	0	0	0	0	0	0

De acordo com as Tabelas 8.1 e 8.2 verifica-se que 100% dos problemas-teste foram reduzidos em pelo menos um vértice e que praticamente todos os problemas-teste só tiveram vértices do conjunto W reduzidos. Apenas em um problema, além de reduzir os vértices do conjunto W, foi transformado um vértice opcional em obrigatório (Regra **R2**).

Com base nas definições de cada regra de redução apresentadas na Seção 5.1.2, justifica-se a não redução de vértices pelas regras **R1**, **R3**, **R4**, **R5** e **R7** da seguinte forma: a regra **R1** não transformou nenhum vértice opcional ($V \setminus T$) em obrigatório (T) pois todos os vértices do conjunto W possuíam mais de um vértice que o cobria. As regras de redução **R3**, **R4** e **R5** não puderam ser aplicadas porque o somatório dos prêmios associados aos vértices obrigatórios era inferior ao prêmio mínimo. Por fim, a regra **R7** não removeu nenhum vértice do conjunto W porque cada vértice desse conjunto não é coberto por todos os vértices do conjunto $V \setminus T$. A regra **R2** foi utilizada pelo fato de o

prêmio do vértice opcional ser necessário para qualquer possível solução atingir o prêmio mínimo que deve ser coletado. Já a regra **R6** foi mais usada porque grande parte dos vértices que precisavam ser cobertos (W) eram cobertos por vértices de T.

Com relação ao fato de os vértices do conjunto W terem sido os mais reduzidos dentre os três tipos de vértices, valem algumas observações. Se todos eles fossem reduzidos, seria preciso se preocupar apenas com a coleta do prêmio mínimo. No caso de se conseguir obter esse prêmio mínimo apenas com os vértices obrigatórios, o problema consistiria apenas na definição da melhor sequência dos vértices de T, isto é, o PRRCP ficaria reduzido ao Problema do Caixeiro Viajante. Contudo, como pode ser observado na Tabela 8.2, em média, o número de vértices do conjunto W reduzido dos problemas-teste dos Grupos 1 e 2 foi de 25,2% e 22,7%, respectivamente. Com isso, pode-se concluir que grande parte dos vértices de W é coberto por vértices opcionais, o que dificulta a princípio a resolução do PRRCP, pois neste caso existem mais combinações de vértices optativos nas soluções viáveis do PRRCP.

A segunda observação é que, mesmo que todos os vértices do conjunto W sejam reduzidos, a explosão combinatória do problema poderia permanecer a mesma, pois os vértices desse conjunto não entram na solução (é preciso apenas que exista algum vértice que esteja a uma distância menor ou igual D de cada vértice de W). Para que a explosão combinatória permaneça a mesma com a remoção de todos os vértices de W, é preciso que o somatório dos prêmios contidos em cada vértice obrigatório não satisfaça ao prêmio mínimo, o que, de fato, acontece em todos os problemas-teste tratados nesse trabalho.

Dessa forma, tem-se indícios de que os problemas-teste desenvolvidos no presente trabalho possuem um grau de dificuldade elevado, pois na adaptação dos problemas-teste do TSPLIB houve uma grande preocupação em se distribuir os vértices de W entre os diferentes tipos de vértices que podem cobrí-los ($V \setminus T$ e T). Além disso, preocupou-se também em distribuir os prêmios de forma que o prêmio mínimo não fosse coletado somente pelos vértices de T. A aplicação das regras de redução também não facilitou muito a resolução do problema, como foi visto pelas Tabelas 8.1 e 8.2.

8.2 Resultados da aplicação da Formulação Matemática

Descreve-se, na Tabela 8.3, os resultados obtidos pela aplicação do otimizador ILOG CPLEX 11.1 à formulação proposta na Seção 4.2.

Tabela 8.3: Resultados da Formulação Matemática

Resultados da Formulação Matemática								
Nome	Limite Superior	Valor Ótimo	Tempo (s)		Nome	Limite Superior	Valor Ótimo	Tempo (s)
g_att48_VT9_T10_W29_25	2291	2291	0,76		g_kroA150_VT30_T30_W90_50	9855,6153	-	8787,93
g_att48_VT9_T29_W10_50	2824	2824	352,81		g_kroA200_VT40_T40_W120_75	15197	15197	15255,90
g_att48_VT16_T16_W16_75	2650	2650	70,06		g_kroB100_VT20_T60_W20_25	18406	18406	7518,73
g_berlin52_VT9_T11_W32_25	4861	4861	0,58		g_kroB100_VT34_T33_W33_50	12557,6271	-	2559,46
g_berlin52_VT16_T18_W18_50	5186	5186	735,34		g_kroB100_VT34_T33_W33_75	14768	14768	470,70
g_berlin52_VT9_T32_W11_75	6916	6916	481,51		g_kroB150_VT30_T30_W90_25	11411,5	-	7166,28
g_bier127_VT24_T77_W26_25	98453	98453	96254,83		g_kroB200_VT40_T40_W120_50	10920,0285	-	5769,76
g_bier127_VT24_T26_W77_50	74745	74745	718,65		g_kroC100_VT20_T60_W20_25	16785	16785	3416,75
g_brazil58_VT18_T20_W20_25	19417	19417	2747,77		g_kroC100_VT34_T33_W33_50	13235	13235	793,09
g_brazil58_VT11_T35_W12_50	23312	23312	262,04		g_kroC100_VT20_T20_W60_75	12582	12582	59,17
g_brazil58_VT11_T12_W35_75	20466	20466	40,55		g_kroD100_VT20_T20_W60_25	10693	10693	114,26
g_brg180_VT36_T36_W108_25	538,7747	-	86390,08		g_kroD100_VT20_T60_W20_50	17375,5186	-	21903,32
g_u159_VT31_T32_W96_50	28613	28613	943,59		g_kroD100_VT34_T33_W33_75	14590	14590	12397,54
g_ch130_VT44_T43_W43_25	4328	-	2011,42		g_kroE100_VT20_T60_W20_25	17199	17199	3454,50
g_ch130_VT26_T26_W78_50	4139	4139	316,68		g_kroE100_VT20_T20_W60_50	10586	10586	443,42
g_ch150_VT30_T30_W90_75	4292	4292	449,17		g_kroE100_VT34_T33_W33_75	23640	23640	12970,58
g_d198_VT39_T40_W119_25	10916	10916	3633,64		g_lin105_VT21_T21_W63_25	8893	8893	65,63
g_eil51_VT17_T17_W17_25	273	273	315,68		g_lin105_VT21_T63_W21_50	11583	11583	2734,49
g_eil51_VT9_T11_W31_50	248	248	0,53		g_lin105_VT35_T35_W35_75	10566	10566	3420,78
g_eil51_VT9_T31_W11_75	348	348	1614,38		g_pr76_VT14_T16_W46_25	66007	66007	8,13
g_eil76_VT14_T46_W16_25	385,1031	-	1120,77		g_pr76_VT24_T26_W26_50	74539	74539	258,87
g_eil76_VT24_T26_W26_50	338	338	340,36		g_pr76_VT14_T46_W16_75	90862	90862	1389,66
g_eil76_VT14_T16_W46_75	332	332	11,06		g_pr107_VT20_T22_W65_50	35869	35869	58,48
g_eil101_VT19_T21_W61_25	356	356	287,15		g_pr107_VT35_T36_W36_75	37292	37292	7691,11
g_eil101_VT19_T61_W21_50	503,5	-	42382,20		g_pr124_VT42_T41_W41_25	35294,5980	-	8893,21
g_eil101_VT33_T34_W34_75	457,0245	-	1554,25		g_pr124_VT24_T25_W75_75	34359	34359	1053,82
g_gr48_VT16_T16_W16_25	3571	3571	11,22		g_pr136_VT26_T28_W82_25	47620	47620	372,94
g_gr48_VT9_T10_W29_50	3524	3524	0,45		g_pr136_VT46_T45_W45_50	58423,7167	-	7238,57
g_gr48_VT9_T29_W10_75	4415	4415	0,47		g_pr144_VT28_T29_W87_50	30294	30294	679,54
g_gr96_VT18_T58_W20_25	42746	42746	964,91		g_pr144_VT48_T48_W48_75	10086,02292	-	7,07
g_gr96_VT32_T32_W32_50	35461	35461	1405,66		g_pr152_VT29_T31_W92_75	49110	49110	21791.03
g_gr96_VT18_T20_W58_75	30015	30015	38,38		g_rat99_VT19_T20_W60_25	792	792	55,83
g_gr120_VT40_T40_W40_25	4301	4301	804,58		g_rat99_VT33_T33_W33_50	847	847	1035,96
g_gr120_VT24_T24_W72_75	4275	4275	234,47		g_rat99_VT19_T60_W20_75	1017	1017	8776,62
g_gr137_VT26_T28_W83_25	39305	39305	686,42		g_rd100_VT34_T33_W33_25	5190,1111	-	62441,29
g_gr137_VT45_T46_W46_50	45547,8676	-	2821,93		g_rd100_VT20_T60_W20_50	6580	-	84662,5
g_hk48_VT9_T29_W10_25	9708	9708	297,35		g_rd100_VT20_T20_W60_75	5421	5421	83,66
g_hk48_VT16_T16_W16_50	7593	7593	146,31		g_si175_VT35_T35_W105_25	4845	4845	738,06
g_hk48_VT16_T16_W16_75	7803	7803	210,07		g_st70_VT14_T14_W42_25	419	419	46,41
g_kroA100_VT20_T20_W60_25	10848	10848	69,62		g_st70_VT14_T42_W14_50	546,6053	-	2651,63
g_kroA100_VT20_T60_W20_50	16544	16544	8595,05		g_st70_VT22_T24_W24_75	505	505	738,65
g_kroA100_VT34_T33_W33_75	14606	14606	1474,86					

Nessa tabela, a primeira coluna mostra o nome do problema-teste; a segunda, um limite inferior para o problema e a terceira, quando encontrado, o valor ótimo obtido. Por fim, a quarta coluna apresenta o tempo de processamento, em segundos.

Com base na Tabela 8.3 verifica-se que, dos 111 problemas-teste do Grupo 1, a formulação proposta neste trabalho conseguiu encontrar o ótimo para 66 problemas e um limite superior para 17 problemas. Para os 28 problemas-teste restantes desse grupo não foi possível aplicar tal formulação, pois o número de variáveis e restrições criadas eram superiores à quantidade de memória principal disponível na máquina. Vale ressaltar que o mesmo problema de insuficiência de memória ocorreu durante a realização do testes com os 33 problemas-teste do Grupo 2.

8.3 Resultados da aplicação das Heurísticas

Esta seção descreve os resultados obtidos pelos algoritmos heurísticos apresentados no Capítulo 6. Foi desenvolvido um Algoritmo Evolutivo e cinco versões do ILS-MRD, cada qual diferenciando no método de geração da solução inicial. A Tabela 8.4 apresenta as seis abordagens desenvolvidas.

Tabela 8.4: Características das heurísticas propostas

Heurísticas propostas			
Nome	Solução Inicial	Busca Local	Reconexão de Caminhos
ILS-MRD_IB	Inserção mais Barata	ILS-MRD	NÃO
ILS-MRD_VP	Vizinho mais Próximo	ILS-MRD	NÃO
ILS-MRD_AD	ADD	ILS-MRD	NÃO
ILS-MRD_DR	DROP	ILS-MRD	NÃO
ILS-MRD_GE	GENIUS	ILS-MRD	NÃO
AE	Todos	ILS-VNRD	SIM

Os testes possuem dois objetivos, sendo o primeiro a comparação entre as diferentes soluções iniciais utilizadas com ILS-MRD no intuito de verificar o impacto da qualidade da solução inicial na obtenção da solução final. Isto é, deseja-se verificar empiricamente se a busca local é ou não dependente de uma boa solução inicial para produzir bons limites superiores. O segundo objetivo é o de comparar duas abordagens diferentes para verificar a eficiência de cada uma delas, sendo uma baseada em busca local e a outra, uma abordagem híbrida baseada nas buscas populacional e local.

As Tabelas 8.5 e 8.6 apresentam os parâmetros utilizados nas diferentes versões do

ILS-MRD e do Algoritmo Evolutivo, respectivamente. Tais parâmetros foram obtidos após uma bateria preliminar de testes.

Tabela 8.5: Parâmetros utilizados no ILS-MRD

ILS-MRD	
Parâmetro	Valor
iterMax_ILS	100
kpMin	5
kpMax	10
δ	2
iterMax_MRD	350

Nessa tabela, *iterMax_ILS* indica o número máximo de iterações executadas pelo ILS-MRD, *kpMin* e *kpMax* representam os graus mínimo e máximo utilizados no mecanismo de perturbação do ILS. O parâmetro δ mostra o grau de incremento do *kpMin* quando o algoritmo atinge *iterMax_ILS* iterações sem melhora. Por fim, o parâmetro *iterMax_MRD* indica o número máximo de iterações do MRD executadas em cada uma das *iterMax_ILS* iterações do ILS.

Tabela 8.6: Parâmetros utilizados no Algoritmo Evolutivo

Algoritmo Evolutivo	
Parâmetro	Valor
maxGeneration	7
populatioSize	5
classBSize	3
classCSize	2
eliteSetSize	5
percDiffSolution	0,15
iterMax_MRD	300
iterMax_ILS	100
iterMax_VNRD	100
kpMin_ILS-VNRD	5
kpMax_ILS-VNRD	7
δ	2

Na Tabela 8.6, o parâmetro *maxGeneration* indica o número máximo de gerações (iterações) do Algoritmo Evolutivo. O parâmetro *populatioSize* apresenta o tamanho da população, isto é, o número de indivíduos criados em cada geração. Os parâmetros *classBSize* e *classCSize* indicam, respectivamente, a quantidade de indivíduos da população que pertencem às classes B e C. A classe A, referente ao conjunto das melhores soluções encontradas em todas as gerações, é representada pelo conjunto elite do algoritmo Reconexão de Caminhos, e seu tamanho é definido pelo parâmetro *eliteSetSize*. Já o parâmetro *per-*

cDiffSolution indica o percentual mínimo de diversidade entre os membros do conjunto elite. O número de iterações do MRD aplicado após a criação do indivíduo é definido pelo parâmetro *iterMax_MRD*. Os parâmetros *iterMax_ILS* e *iterMax_VNRD* indicam, respectivamente, o número máximo de iterações do ILS e do VNRD em cada iteração do ILS. Por fim, *kpMin_ILS-VNRD* indica o grau mínimo de perturbação, *kpMax_ILS-VNRD* indica o grau máximo de perturbação e δ mostra o incremento dado ao grau de perturbação a cada *iterMax_ILS* iterações.

8.3.1 Comparação entre as heurísticas construtivas

Esta seção apresenta os resultados das comparações realizadas entre as heurísticas utilizadas para geração da solução inicial.

As Tabelas 8.7 e 8.8 apresentam a comparação entre os algoritmos de geração da solução inicial para os problemas-teste do Grupo 1 e do Grupo 2, respectivamente. Na primeira coluna tem-se o nome do algoritmo; na segunda, o custo médio obtido por cada algoritmo; na terceira, o desvio médio do custo das soluções em relação ao melhor custo conhecido; na quarta, o tempo médio de cada execução em segundos e, por fim, na quinta coluna indica-se a taxa de sucesso de cada algoritmo em cada grupo de problemas-teste. Essa taxa corresponde à porcentagem de problemas-testes em que o algoritmo conseguiu encontrar a melhor solução conhecida em pelo menos uma das dez execuções.

O desvio médio é calculado pela equação (8.1):

$$Desvio = \frac{Media - MelhorValor}{MelhorValor} \times 100 \tag{8.1}$$

Os experimentos foram delimitados por um número máximo de iterações do algoritmo e não por um tempo máximo. A justificativa para tal escolha se deve ao fato de que, por ser um problema novo na literatura, definiu-se como prioridade encontrar bons limites superiores para os mesmos, sem se preocupar muito com o tempo. Portanto, a média de tempo apresentada nas tabelas diz respeito ao tempo total gasto pelo algoritmo para executar todas as iterações, não significando que o valor final da solução de cada execução tenha sido encontrado no fim do tempo de execução.

Com base nas Tabelas 8.7 e 8.8, verifica-se que os métodos de geração da solução inicial tiveram um comportamento semelhante para os dois grupos. A heurística GENIUS foi a que obteve as melhores soluções iniciais para o PRRCP. Em contrapartida, a heurística do Vizinho mais Próximo foi a que gerou soluções de pior qualidade dentre todas as

Tabela 8.7: Comparação entre os algoritmos construtivos aplicados aos problemas-teste do **Grupo 1**

Problema-testes do **Grupo 1**				
Heurística	Custo Médio	Desvio Médio (%)	Tempo médio (s)	Sucesso (%)
Inserção mais Barata	28359,14	19,41	0	0
Vizinho mais Próximo	152855,4	81,7	0	0
ADD	56110,3	59,52	0,23	0
DROP	54450,6	59,05	0,24	0
GENIUS	**21347,2**	**8,29**	**4,61**	**4,5**

Tabela 8.8: Comparação entre os algoritmos construtivos aplicados aos problemas-teste do **Grupo 2**

Problemas-teste do **Grupo 2**				
Heurística	Custo Médio	Desvio Médio (%)	Tempo médio (s)	Sucesso (%)
Inserção mais Barata	41957,9	21,01	0,02	0
Vizinho mais Próximo	259688,8	86,2	1,33	0
ADD	130732,3	74,12	2,9	0
DROP	129985,2	74,14	2,53	0
GENIUS	**38002**	**11,54**	**14,5**	**4,5**

heurísticas. Com exceção da heurística GENIUS, o tempo de execução para criação de uma solução inicial foi baixo.

Uma possível justificativa do resultado da heurística GENIUS ter sido a que obteve os melhores resultados se deve à natureza dos procedimentos de inserção e remoção de vértices na solução que está sendo construída. Como apresentado na Subseção 6.2.5, o vértice que está sendo inserido na solução é colocado entre dois vértices mais próximos a ele que já fazem parte da solução, independentemente de eles serem adjacentes ou não. Além disso, para cada inserção de um novo vértice na solução, são testadas diferentes posições de inserção. Dessa forma, são destacadas as seguintes vantagens observadas na heurística GENIUS em relação às demais: (1) a GENIUS possui um método mais elaborado de inserção dos vértices no qual diversas posições entre diferentes vértices são testadas em uma mesma iteração, de acordo com o conjunto N_p do mesmo; (2) os vértices adjacentes ao que está sendo inserido não precisam necessariamente serem adjacentes na solução, pois a heurística possui mecanismos eficientes de rearranjo das posições dos demais vértices, de forma que eles não se distanciem dos mais próximos a eles; (3) na segunda etapa da heurística, uma vez que todos os vértices já foram inseridos, o processo de remoção e de re-inserção tem a oportunidade de posicionar os vértices nas suas posições ótimas ou pelo menos em posições que já estejam próximas à ótima. Como pode ser observado nas

Tabelas 8.7 e 8.8, para alguns problemas-teste, a heurística GENIUS encontrou a solução ótima já na fase de geração da solução inicial.

O fato de a heurística GENIUS possuir um tempo maior em relação às demais heurísticas se justifica por esta possuir mais fases que as demais. A heurística GENIUS possui duas fases, sendo a primeira de inserção e a segunda de remoção e inserção novamente dos vértices. A fase de inserção e remoção possuem dois tipos de inserção e remoção respectivamente, sendo que são testadas diferentes posições para inserção e diferentes arranjos dos vértices remanescentes na remoção. Então, era de se esperar que tal heurística gastasse mais tempo do que as demais, mas que, por outro lado, é compensado pela qualidade da solução gerada.

Em relação à heurística que obteve o pior desempenho, a do Vizinho mais Próximo, pode-se dizer que esse desempenho se deve principalmente à visão local que o procedimento possui, isto é, o procedimento analisa apenas o último vértice inserido para tomar a decisão de qual será o próximo a ser inserido na solução. Dessa forma, no início do procedimento é criada uma solução de custo relativamente baixo. Mas em determinado ponto, só restam vértices que estão distantes entre si para serem inseridos na solução. Com isso, o custo da solução se eleva, gerando soluções iniciais ruins em relação ao custo.

Em se tratando dos demais métodos de geração da solução inicial, verifica-se que as heurísticas ADD e DROP tiveram um desempenho bastante semelhante. Essa semelhança pode ser justificada através de dois pontos de vista diferentes. O primeiro é que heurística ADD constitui a primeira etapa da heurística DROP e, assim, ambos seguem o mesmo conceito de inserção. O segundo ponto de vista diz respeito ao cálculo do benefício de se inserir ou remover determinado vértice. Esses dois cálculos são realizados com base nos mesmos fatores, a saber, na distância total da rota, no prêmio total da rota e na cobertura dos vértices do conjunto W.

A heurística de Inserção mais Barata também apresentou um desempenho satisfatório, ficando atrás apenas das soluções geradas pela heurística GENIUS. A grande diferença entre as heurísticas que faz com que a heurística GENIUS se sobressaia, se refere à maneira de inserção do vértice. Na heurística da Inserção mais Barata, o vértice que está sendo inserido também é colocado na posição adjacente aos vértices mais próximos a ele, mas, nesse procedimento, não existe nenhum mecanismo de rearranjo da solução após a inserção de um vértice, nem a segunda fase de remoção e inserção novamente dos vértices. Com isso, o vértice pode ficar numa posição longe da melhor posição que ele deveria ficar, já que a escolha da ordem de inserção dos vértices é definida aleatoriamente.

Portanto, com base nas Tabelas 8.7 e 8.8, conclui-se que, para os problemas-teste considerados, os melhores métodos de geração da solução inicial foram as heurísticas que possuem uma visão global da solução que está sendo construída e que só se preocupam com a distância entre os vértices para definir a posição de inserção, deixando que a restrição do prêmio mínimo a ser coletado seja contemplada na fase de busca local.

8.3.2 Comparação das Versões do ILS-MRD

Esta seção apresenta uma comparação dos resultados encontrados pelas diferentes versões do ILS-MRD. O objetivo das comparações é verificar a influência da qualidade da solução inicial na obtenção de boas soluções na fase de busca local.

As Tabelas 8.9 e 8.10 resumem os resultados dos testes realizados com os problemas-teste do Grupo 1 e do Grupo 2, respectivamente. Nessas tabelas, a primeira coluna indica o nome do algoritmo; a segunda, o custo médio de todos os problemas-teste; a terceira, o desvio médio do custo das soluções em relação ao melhor custo conhecido e a quarta, o tempo médio gasto em cada execução em segundos. A penúltima coluna indica a taxa de sucesso do algoritmo em encontrar a melhor solução conhecida em pelo menos uma das dez execuções , enquanto a última mostra a porcentagem de problemas-teste em que cada algoritmo encontrou a melhor solução sozinho.

Tabela 8.9: Comparação dos resultados encontrados por diferentes versões do ILS-MRD em problemas do **Grupo 1**

Problemas-teste do **Grupo 1**					
Versão	Custo	Desvio (%)	Tempo (s)	Sucesso (%)	Sozinho (%)
`ILS-MRD_IB`	20090,5	0,09	1136,9	95,5	0,9
`ILS-MRD_VP`	20085,9	0,08	1031,9	95,5	0
ILS-MRD_AD	**20080**	**0,06**	**1255,9**	**96,4**	**0**
`ILS-MRD_DR`	20081,4	0,06	1226,9	96,4	0,9
`ILS-MRD_GE`	20081,8	0,06	1191,2	96,4	0

De acordo com a Tabela 8.9, verifica-se que todas as versões tiveram desempenho semelhante nos problemas-teste do Grupo 1, mas o algoritmo `ILS-MRD_AD` foi o que obteve o melhor conjunto de resultados. Isso se deve ao fato de que seu desvio médio foi o menor, no caso, de 0,06% e, além disso, obteve, em pelo menos uma das execuções, o melhor valor conhecido em 96,4% dos problemas-teste do Grupo 1. Já os algoritmos `ILS-MRD_IB` e `ILS-MRD_DR` , encontraram cada um, o melhor limite superior para 0,9% dos problemas desse grupo.

Com base na Tabela 8.9, verifica-se que, em geral, as soluções encontradas por essas versões do ILS-MRD são de boa qualidade, pois todos eles encontraram o valor ótimo ou o melhor limite superior em mais de 95% do total dos problemas-teste do Grupo 1, com um desvio médio abaixo de 0,1%, mostrando boa convergência empírica.

Tabela 8.10: Comparação dos resultados encontrados por diferentes versões do ILS-MRD em problemas do **Grupo 2**

Problemas-teste do **Grupo 2**					
Versão	Custo	Desvio (%)	Tempo (s)	Sucesso (%)	Sozinho (%)
ILS-MRD_IB	**33752,7**	**0,34**	**4897,5**	**76,7**	**3,3**
ILS-MRD_VP	33795,9	0,44	4761	80,0	0
ILS-MRD_AD	33813,5	0,47	5093.2	73,3	0
ILS-MRD_DR	33802,9	0,45	4513.8	70,0	0
ILS-MRD_GE	33776,3	0,38	4887.4	76,7	0

Com base na Tabela 8.10, verifica-se também que todas as versões do ILS-MRD obtiveram um desempenho semelhante nos problemas-teste do Grupo 2. Entretanto, o algoritmo ILS-MRD_IB se sobressaiu pelas seguintes características: (1) foi o que obteve o melhor conjunto de resultados, pois o seu desvio médio foi o menor, no caso, de 0,34%; (2) sua taxa de sucesso foi a segunda maior, no caso, de 76,7% e (3) foi o único a encontrar isoladamente o melhor valor para 3,3% dos problemas-teste. Observa-se, ainda, que o algoritmo ILS-MRD_VP encontrou o melhor valor conhecido para 80% dos problemas do Grupo 2.

Em relação ao tempo médio de execução, verifica-se que todas as versões tiveram tempos de processamento semelhantes nos dois grupos. Um ponto importante a mencionar diz respeito ao tempo de execução do algoritmo ILS-MRD_GE. Como a heurística GENIUS foi a que demorou mais tempo para gerar uma solução inicial, podia-se esperar que, pela qualidade da solução inicial, o algoritmo gastaria um tempo menor que a média para as execuções do ILS-MRD. Mas, de acordo com Tabela 8.9, o tempo de execução do algoritmo ILS-MRD_GE foi semelhante aos tempos de execução das demais versões. Uma possível justificativa para esse fato é que, como a solução inicial gerado pelo GENIUS já possui uma qualidade muito boa, o algoritmo ILS-MRD gasta muito tempo trabalhando com soluções de custo superior à inicial, pois grande parte dos vértices já se encontra em suas posições ótimas. Dessa forma, não é qualquer movimento na solução que traz alguma melhora para o custo, o que faz com que grande parte do tempo de execução do algoritmo seja gasto com um esforço em vão.

Ainda em relação à versão ILS-MRD_GE, esperava-se que essa abordagem se destacasse

entre as demais, já que suas soluções iniciais eram melhores que as das demais versões. Entretanto, após os testes, verificou-se que esta hipótese não pôde ser validada. A justificativa que se tem para que tal hipótese seja falsa é que a fase de busca local possui vários movimentos que foram criados com base nas duas fases da heurística GENIUS. Sendo assim, as soluções iniciais submetidas à fase de busca local já estavam bem formadas, sendo difícil encontrar algum movimento que conseguisse melhorá-las.

Por outro lado, verifica-se que as estruturas de vizinhança criadas com base na heurística GENIUS foram de grande importância para a fase de busca local reduzir o custo das soluções. De acordo com as Tabelas 8.9 e 8.10, percebe-se que todas as versões encontraram soluções de boa qualidade para a maioria dos problemas-teste. Com isso, conclui-se que o ILS-MRD proposto no presente trabalho não é dependente de soluções iniciais de boa qualidade, pois ele possui estruturas de vizinhança eficientes para promover a convergência dessas para soluções finais ainda melhores.

8.3.3 Comparação entre o ILS-MRD e o Algoritmo Evolutivo

Esta seção apresenta uma comparação entre os resultados obtidos pelas melhores versões do ILS-MRD em cada grupo (`ILS-MRD_AD` e `ILS-MRD_IB`, respectivamente) com aqueles produzidos pelo Algoritmo Evolutivo. O objetivo dessas comparações é verificar qual abordagem (apenas busca local ou híbrida com busca populacional e local) é mais propensa a resultar em boas soluções para um maior número de problemas-teste ou se ambas apresentam um comportamento semelhante.

As Tabelas 8.11 e 8.12 resumem os resultados das comparações envolvendo problemas-teste do Grupo 1 e Grupo 2, respectivamente. Nessas tabelas, a primeira coluna indica o nome do algoritmo; a segunda, o custo médio de todos os problemas; a terceira, o desvio médio do custo das soluções em relação ao melhor custo conhecido e a quarta, o tempo médio gasto em cada execução do algoritmo. Na penúltima coluna mostra-se a taxa de sucesso de cada algoritmo em alcançar o melhor custo em pelo menos uma das dez execuções. Por fim, na última coluna, apresenta-se a porcentagem de problemas que cada algoritmo encontrou o menor custo sozinho.

O desvio médio é calculado pela equação (8.2):

$$Desvio = \frac{Media - MelhorValor}{MelhorValor} \times 100 \tag{8.2}$$

De acordo com a Tabela 8.11, observa-se que o Algoritmo Evolutivo (AE) obteve

Tabela 8.11: Comparação de desempenho entre ILS-MRD_*AD* e Algoritmo Evolutivo em problemas-teste do **Grupo 1**

Problemas-teste do **Grupo 1**					
Versão	Custo	Desvio (%)	Tempo (s)	Sucesso (%)	Sozinho (%)
`ILS-MRD_AD`	20080	0,06	1255,9	96,4	0
AE	**20073,5**	**0,02**	**1538,3**	**99,1**	**1,8**

um desempenho superior ao de `ILS-MRD_AD`. Isto se deve ao fato de que o desvio médio das soluções de AE foi inferior ao de `ILS-MRD_AD` , no caso, de 0,02% e, além disso, o Algoritmo Evolutivo encontrou o melhor valor conhecido em 99,1% dos casos do Grupo 1, e em 1,8% dos problemas-teste encontrou esse melhor valor sozinho.

Em relação aos tempos de execução, verifica-se que estes mostraram-se próximos, apesar de o Algoritmo Evolutivo possuir o procedimento Reconexão por Caminhos adicional em relação às outras propostas.

O motivo de a Reconexão por Caminhos não ter elevado muito o tempo de execução do Algoritmo Evolutivo é justificado pelo fato de o número de vértices nos problemas-teste do Grupo 1 não ser muito elevado e, dessa forma, pequenos esforços do algoritmo já melhoram bastante a qualidade da solução. No Algoritmo Evolutivo, a população é gerada pelos diferentes métodos de geração da solução inicial; com isso, os custos e, consequentemente, o arranjo dos vértices nas soluções é o mais variado possível. Com uma leve busca local realizada em cada indivíduo da população após sua criação, as soluções são bem melhoradas, formando-se uma população com uma qualidade melhor. Então, quando essas soluções são encaminhadas para a Reconexão por Caminhos, poucos movimentos precisam ser efetuados para se sair da solução base e alcançar a solução guia, o que faz com que o tempo de execução do algoritmo não cresça muito.

Tabela 8.12: Comparação de desempenho entre ILS-MRD_*IB* e Algoritmo Evolutivo em problemas-teste do **Grupo 2**

Problemas-teste do **Grupo 2**					
Versão	Custo	Desvio (%)	Tempo (s)	Sucesso (%)	Sozinho (%)
`ILS-MRD_IB`	33752,7	0,34	4897,5	76,7	3.3
AE	**33641,2**	**0,14**	**9294**	**91,0**	**16,7**

Com base na Tabela 8.12, observa-se que a diferença do desempenho do Algoritmo Evolutivo em relação ao `ILS-MRD_IB` foi mais significativa para os problemas-teste do Grupo 2. De fato, o Algoritmo Evolutivo apresentou um desvio médio de 0,14%, inferior

ao do `ILS-MRD_IB` , que foi de 0,34%. Além disso, o número de problemas-teste que o Algoritmo Evolutivo encontrou o melhor valor conhecido, seja esse valor o ótimo ou um limite superior, foi superior a 90%, valor esse maior do que o alcançado pelo algoritmo `ILS-MRD_IB`, de 76,7%. Para os problemas-teste do Grupo 2, o Algoritmo Evolutivo encontrou, sozinho, a melhor solução conhecida para aproximadamente 16,7% dos casos. Assim, tal como no caso anterior, também pode-se concluir que o AE não necessita de soluções iniciais de boa qualidade para convergir para boas soluções finais.

Em relação ao tempo de execução, verifica-se que o Algoritmo Evolutivo gastou em média o dobro de tempo que a outra abordagem proposta. Tal fato é justificado, ao contrário dos problemas-teste do Grupo 1, pela utilização da Reconexão por Caminhos. Nos problemas do Grupo 2, o conjunto de vértices é muito maior; com isso, a busca local realizada após a criação de cada indivíduo não melhora tanto a qualidade da solução como acontece nos problemas-teste do Grupo 1. Dessa forma, a distância da solução base em relação guia é maior, o que faz com que a Reconexão por Caminhos demande mais tempo para ser executada. Adicionalmente, como pode ser observado na Tabela 8.10, o tempo de execução para o Grupo 2 já é naturalmente mais elevado.

De acordo com os resultados, observou-se que o maior benefício em se utilizar a Reconexão por Caminhos foi justamente a de melhorar a qualidade da solução que é passada ao ILS-VNRD, facilitando a convergência na fase de busca local.

8.3.4 Análise de probabilidade empírica

No próximo conjunto de experimentos, verificou-se a distribuição de probabilidade empírica de alcançar um dado valor alvo (ou seja, encontrar uma solução com um custo no mínimo tão bom quanto o valor alvo) em função do tempo para dois problemas-teste, um do Grupo 1 e outro do Grupo 2. No caso do Grupo 1, o alvo escolhido foi o valor ótimo; já no caso do Grupo 2, foram considerados dois alvos, sendo um deles difícil e outro médio. Os experimentos foram conduzidos conforme a proposta de [Aiex *et al.*, 2002]. Os resultados foram plotados associando-se ao i-ésimo tempo de execução a probabilidade $p_i = (i - \frac{1}{2})/100$, gerando-se pontos $z_i = (t_i, p_i)$, para $i = 1, \ldots, 100$.

A Figura 8.1 apresenta os resultados comparando todas as versões do ILS-MRD e o Algoritmo Evolutivo, considerando o valor alvo igual a 32792, correspondente ao valor ótimo do problema-teste `pr107_VT35_T36_W36_75`.

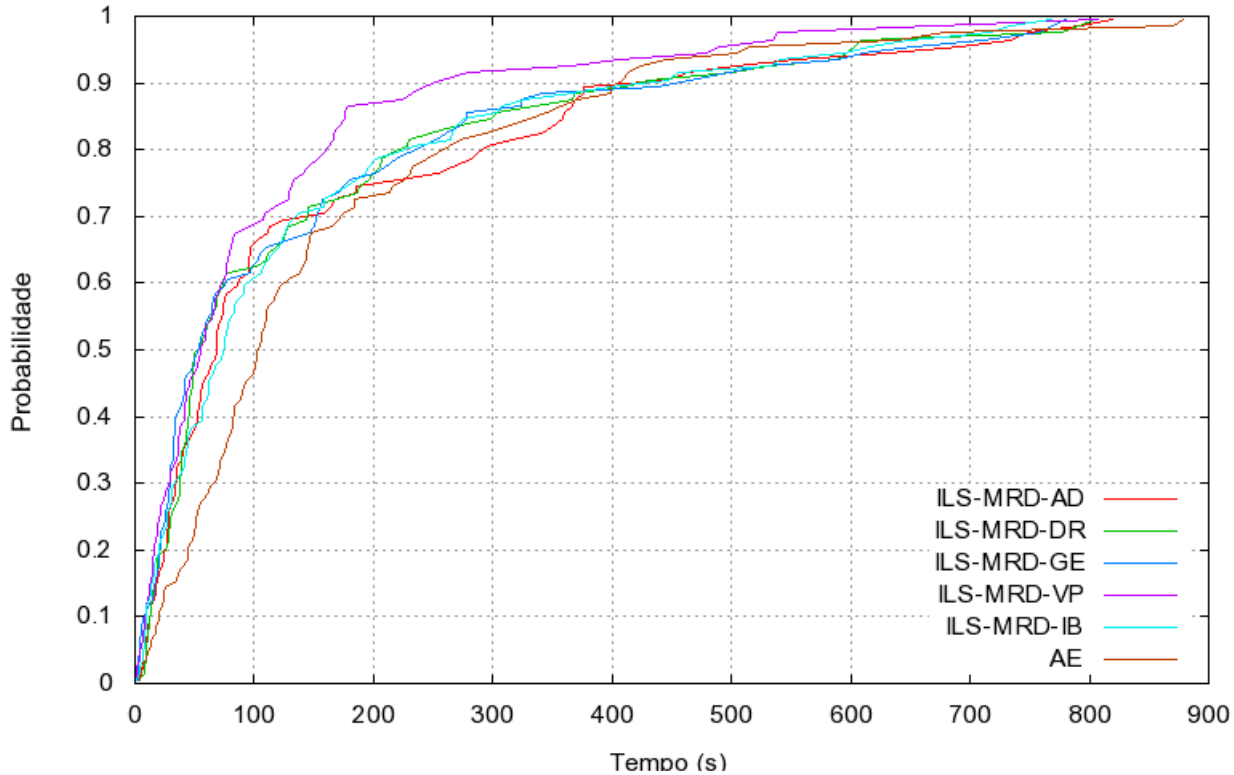

Figura 8.1: Distribuição de probabilidade empírica relativa ao problema `pr107_VT35_T36_W36_75`, alvo 32792

Pode-se verificar pela Figura 8.1 que o Algoritmo Evolutivo alcança o valor alvo, com quase 100% de probabilidade, em um tempo maior que todas as demais versões do ILS-MRD. Entre as versões do ILS-MRD, observa-se que todas elas demandam cerca de 800 segundos de processamento para alcançar o valor alvo com quase 100% de probabilidade. Entretanto, como regra geral, fixando-se um tempo de processamento, a versão `ILS-MRD_VP` é a que tem a maior probabilidade de alcançar o valor alvo.

A Figura 8.2 compara as versões do ILS-MRD e o Algoritmo Evolutivo usando um alvo de valor 61428, relativo ao problema-teste `gr229_VT45_T46_W138_25`. Por ser o terceiro melhor valor frequentemente alcançado por todos os algoritmos, é considerado um alvo médio.

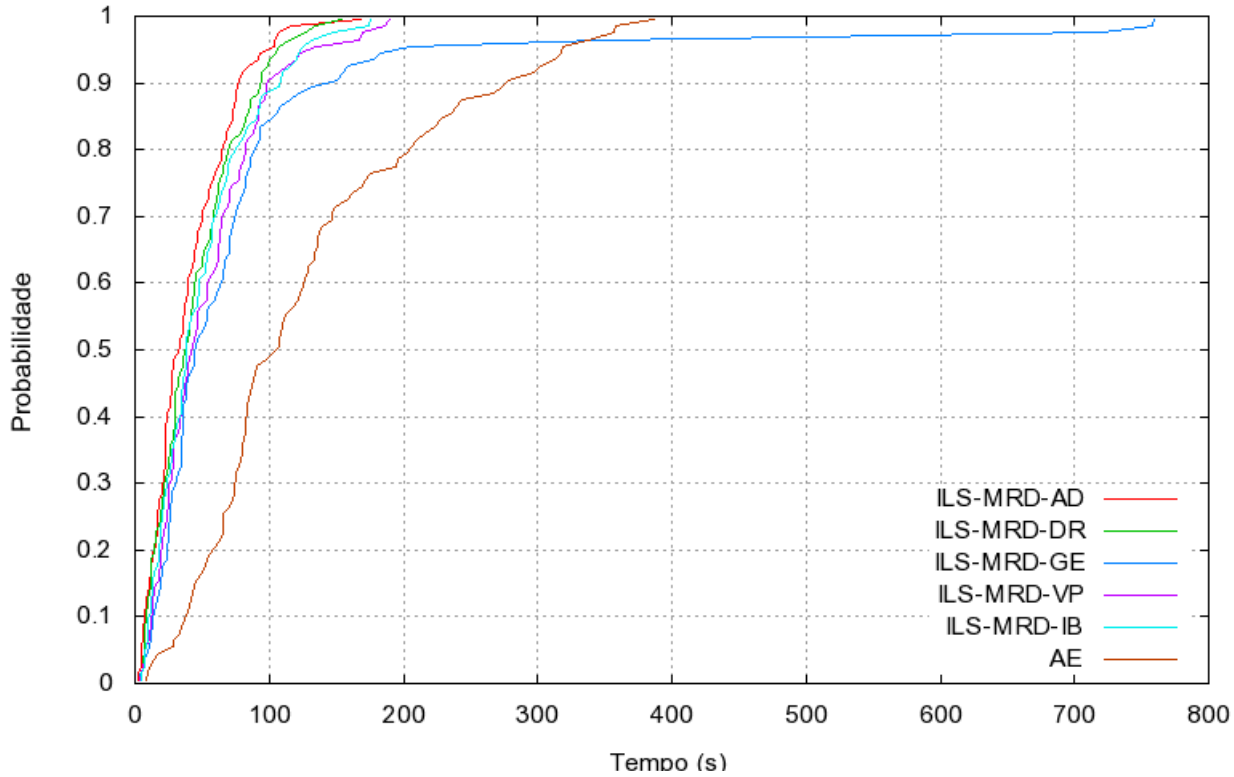

Figura 8.2: Distribuição de probabilidade empírica relativa ao problema `gr229_VT45_T46_W138_25`, alvo 61428

Observa-se pela Figura 8.2 que, com exceção da versão `ILS-MRD_GE`, as versões do ILS-MRD demandam cerca de 200 segundos de processamento para alcançar o valor alvo com quase 100% de probabilidade. A versão `ILS-MRD_GE` possui um comportamento semelhante ao das outras versões do ILS-MRD, porém, para algumas execuções, o alvo foi alcançado em um tempo compreendido entre 200 e 800 segundos. Como regra geral, fixando-se um tempo de processamento, a versão `ILS-MRD_AD` é a que tem a maior probabilidade de alcançar o valor alvo. Em relação ao Algoritmo Evolutivo, verifica-se que este é o que tem a menor probabilidade de alcançar o valor alvo em um dado tempo de processamento. A exceção ocorre apenas com relação à versão `ILS-MRD_GE`, a qual a partir de aproximadamente 350 segundos tem probabilidade menor de alcançar o alvo.

A Figura 8.3 apresenta os resultados comparando todas as versões do ILS-MRD e o Algoritmo Evolutivo, considerando um alvo difícil, de valor igual a 61278, correspondente ao melhor valor encontrado para o problema-teste `gr229_VT45_T46_W138_25`.

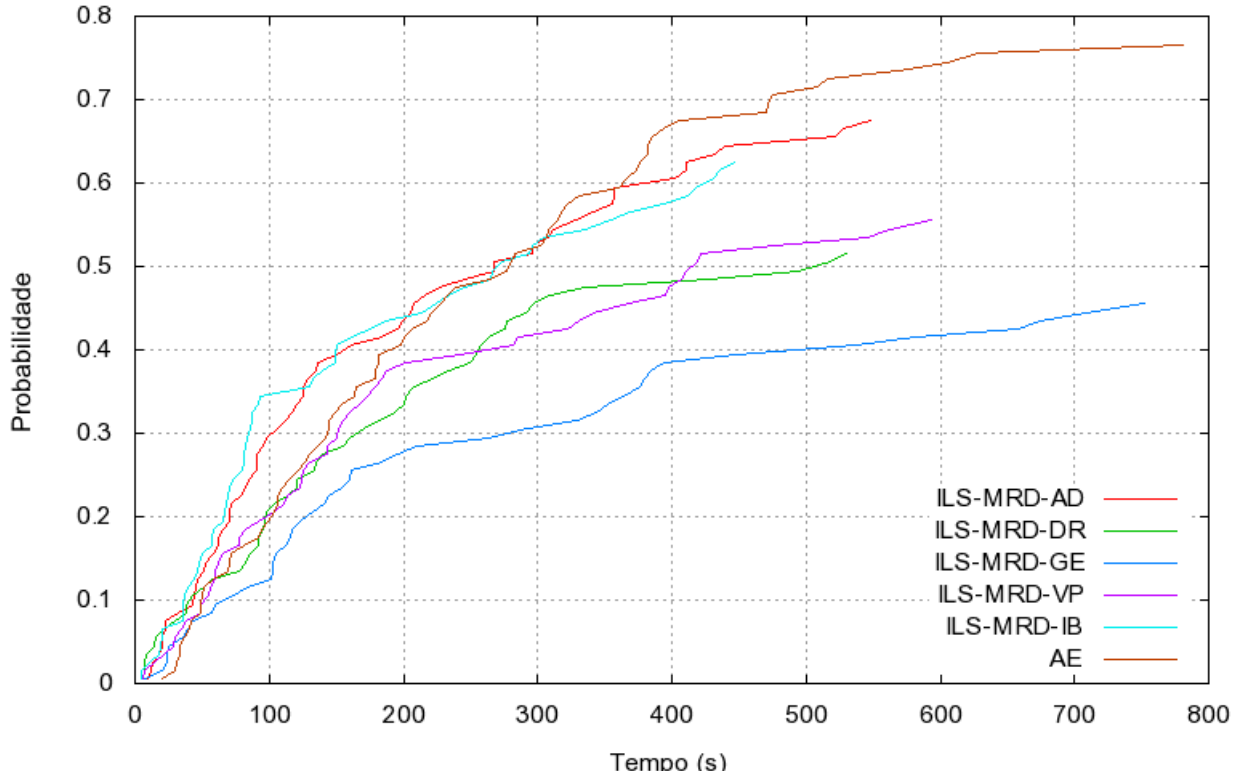

Figura 8.3: Distribuição de probabilidade empírica relativa ao problema `gr229_VT45_T46_W138_25`, alvo 61278

Verifica-se pela Figura 8.3 que, nenhuma das abordagens propostas alcança o alvo em 100% das execuções. O Algoritmo Evolutivo é o que encontra com mais frequência o alvo, com quase 80% de probabilidade. A versão `ILS-MRD_GE` foi a que menos encontrou o alvo, em menos de 50% das execuções ela obteve tal êxito. As demais versões do ILS-MRD, para o alvo em questão, possuem mais de 50% de probabilidade de alcançá-lo. Em relação ao tempo de excução, observa-se que, com exceção do Algoritmo Evolutivo e a versão `ILS-MRD_GE`, as outras versões do ILS-MRD demandam cerca de 600 segundos para alcançar o alvo. O Algoritmo Evolutivo e a versão `ILS-MRD_GE`, para algumas execuções, gastam mais de 700 segundos para alcançá-lo.

O fato de o Algoritmo Evolutivo demandar em algumas execuções um tempo maior que as demais versões pode ser justificado pela utilização do mecanismo de Reconexão por Caminhos. Dependendo da diferença simétrica das soluções base e guia, o tempo de execução pode aumentar consideravelmente.

8.3.5 Considerações finais

De acordo com os resultados apresentados, o Algoritmo Evolutivo foi o que gerou as melhores soluções para a maioria dos problemas-teste. Esse mérito se deve à característica do algoritmo em trabalhar com os dois tipos de buscas, local e populacional, mesclando soluções de boa qualidade com soluções de qualidade inferior, fazendo com que a busca local tenha mais chances de melhorar a solução. O bom desempenho do Evolutivo se deve à idéia que está por trás de sua busca local realizada pelo algoritmo ILS-VNRD, que contém estruturas muito semelhantes às técnicas de inserção e remoção da heurística GENIUS, que se mostraram eficientes. Por exemplo, a segunda fase da GENIUS consiste em remover e reinserir os vértices da solução contruída na primeira fase. Na busca local feita pelo ILS-VNRD, existe uma vizinhança para remover um vértice e reinserí-lo na solução. Quando essa busca está trabalhando nessa vizinhança, são realizadas várias iterações nela antes de se mudar para outra. É nesse ponto que o conceito do GENIUS e do ILS-VNRD se assemelham e se diferem do ILS-MRD. Neste último, existe a chance de se efetuar vários movimentos na mesma vizinhança, mas a chance de isso acontecer é baixa, já que a escolha da vizinhança utilizada em cada iteração é feita de forma aleatória.

Do exposto, conclui-se que a abordagem híbrida composta das buscas populacional e local teve um desempenho melhor para o PRRCP do que a abordagem baseada apenas em busca local.

Capítulo 9

Conclusões e Trabalhos Futuros

Este trabalho abordou uma generalização do Problema do Caixeiro Viajante (PCV), denominado Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP).

Por ser um problema novo na literatura, com apenas um trabalho encontrado, foi desenvolvido um gerador de instâncias que adapta para o PRRCP, problemas-teste relativos ao Problema do Caixeiro Viajante disponíveis na literatura. Esse gerador foi concebido de forma que houvesse uma maior distribuição de cobertura dos vértices que deviam ser cobertos (conjunto W) entre os vértices obrigatórios T e opcionais ($V \setminus T$). Além disso, evitou-se que o prêmio mínimo fosse satisfeito apenas com os vértices obrigatórios, situação que transformaria o PRRCP no PCV. Foram gerados dois grupos de problemas-teste, o primeiro envolvendo até 200 vértices e o segundo, de 201 a 400 vértices.

Na tentativa de reduzir o número de vértices de cada problema-teste, foram desenvolvidas três novas regras de redução, sendo estas aplicadas ao problema juntamente com outras quatro encontradas na literatura. A aplicação dessas regras não conseguiu reduzir muitos vértices, o que era de se esperar pela própria idéia construtiva do gerador. Em média, 20% dos vértices pertencentes ao conjunto W foram eliminados por uma única regra de redução. Com base nesse fato, conclui-se que os problemas-teste gerados podem ser considerados difíceis de serem resolvidos, pois mesmo com o conjunto W sendo parcialmente reduzido, a explosão combinatória do problema permaneceu a mesma.

Desenvolveu-se uma nova formulação matemática para o PRRCP, com número de variáveis e restrições da ordem de $O(n^3)$. Com base nessa formulação, nem todos os problemas-teste puderam ser testados, pois a máquina utilizada não possuia memória suficiente para gerar o modelo. Além disso, para alguns problemas, só foi possível encontrar limites superiores. Para 64 problemas-teste do Grupo 1, a formulação encontrou o valor ótimo

do problema, ajudando assim na validação dos procedimentos heurísticos propostos.

Sendo o PRRCP uma generalização do PCV, o qual é NP-difícil, foram propostos seis algoritmos heurísticos para encontrar soluções subótimas do PRRCP. Dentre esses, um é baseado em Algoritmos Evolutivos e os outros cinco são versões baseadas na metaheurística *Iterated Local Search* (ILS), tendo o Método Randômico de Descida (MRD) como mecanismo de busca local. Para gerar soluções iniciais para esses algoritmos, foram adaptados cinco procedimentos heurísticos clássicos do PCV. Resultados computacionais mostraram que o procedimento GENIUS foi o que gerou as melhores soluções iniciais, com um desvio médio de 8,9% para os problemas-teste do Grupo 1 e de 11,54% para os do Grupo 2. Além disso, em alguns problemas, este procedimento conseguiu encontrar a melhor solução conhecida. O procedimento Inserção Mais Barata foi o segundo melhor, sendo capaz de produzir soluções iniciais com desvio médio de 19,41% em problemas do Grupo 1 e 21,01% em problemas do Grupo 2. Os demais procedimentos geravam soluções com desvio médio superior a 50%. Observou-se que tanto o GENIUS quanto a Inserção Mais Barata se sobressaíram por possuírem uma melhor visão global da solução que está sendo construída e levar em consideração apenas as distâncias no processo construtivo, deixando que a restrição de prêmio mínimo fosse contemplada na fase de busca local.

Para explorar o espaço de soluções, foram criadas diferentes estruturas de vizinhança para o PRRCP, baseadas nas fases de inserção e remoção da heurística GENIUS. Essas estruturas de vizinhança se mostraram bastante eficientes para a convergência das soluções iniciais em soluções finais de qualidade.

As cinco versões dos algoritmos baseados em ILS-MRD se diferenciavam pelo método de geração da solução inicial. Resultados computacionais mostraram que a fase de busca local não é, aparentemente, dependente de uma boa solução inicial para nenhuma das versões do ILS-MRD. Em média, o desvio foi inferior a 0,1% nos problemas-teste do Grupo 1 e inferior a 0,5% nos problemas do Grupo 2. Observou-se que o grande mérito da busca local está nas estruturas de vizinhança desenvolvidas neste trabalho com base na heurística GENIUS.

O Algoritmo Evolutivo desenvolvido faz uso de um procedimento baseado na metaheurística *Iterated Local Search* (ILS), combinado com a Descida Randômica em Vizinhança Variável (VNRD) como método de busca local e aplica a estratégia de intensificação Reconexão por Caminhos periodicamente. O seu desempenho foi comparado com as versões do algoritmo de busca local ILS-MRD que obtiveram os melhores resultados. Experimentos computacionais mostraram que o Algoritmo Evolutivo obteve um desem-

penho superior ao do ILS-MRD para os dois grupos de problemas-teste. Para o Grupo 1, o desvio médio desse algoritmo foi de 0,02% e para o Grupo 2, de 0,14%, enquanto que o menor desvio apresentado pelas versões do ILS-MRD para os problemas do Grupo 1 e do Grupo 2 foram de 0,06% e 0,34%, respectivamente. Observou-se que, tal como o ILS-MRD, o Algoritmo Evolutivo também não foi dependente de boas soluções iniciais e que a abordagem híbrida composta das buscas populacional e local teve desempenho superior à baseada apenas em busca local.

Como trabalhos futuros, propõe-se um estudo aprofundado de técnicas exatas para resolução do PRRCP, como, por exemplo, geração de cortes e geração de colunas, já que muitos problemas-teste propostos neste trabalho possuem apenas um limite superior. A justificativa de se propor tais técnicas exatas se deve ao fato de que estas já foram amplamente estudadas para o PCV e se mostraram bem sucedidas. Como o PRRCP é uma generalização do PCV, acredita-se que as mesmas sejam promissoras para resolução do PRRCP. Outra proposta consiste na otimização do desempenho da avaliação das soluções. Como os movimentos usados são baseados na heurística GENIUS, seria interessante desenvolver uma técnica que calculasse o novo custo da solução sem reavaliar toda a solução, reduzindo o tempo demandado pelo algoritmo. Propõe-se, também, o desenvolvimento de novas regras de redução para o PRRCP que não levem em consideração a necessidade de se ter coletado o prêmio mínimo.

Referências

[Aiex *et al.*, 2002] Aiex, R. M., Resende, M. G. C., e Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: an experimental investigation. *Journal of Heuristics*, 8:343–373.

[Arkin e Hassin, 1994] Arkin, E. M. e Hassin, R. (1994). Approximating algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218.

[Balas, 1989] Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, 19:621–636.

[Balas e Ho, 1980] Balas, E. e Ho, A. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimizations: A computational study. *Mathematical Programming*, 12:37–70.

[Claus, 1984] Claus, A. (1984). A new formulation for the traveling salesman problem. *SIAM J. Alg. Disc. Math.*, 5:21–25.

[Current, 1981] Current, J. R. (1981). *Multiobjective Design of Transportation Networks*. Tese de Doutorado, The Johns Hopkins University.

[Current *et al.*, 1984] Current, J. R., Revelle, C., e Cohon, J. (1984). The shortest covering path problem: An application of locational constraints to network design. *Journal of Regional Science*, 24:161–183.

[Current e Rolland, 1994] Current, J. R. e Rolland, E. (1994). Efficient algorithms for solving the shortest covering path problem. *Transportation Science*, 28:317–327.

[Current e Schilling, 1989] Current, J. R. e Schilling, D. A. (1989). The covering salesman problem. *Transportation Science*, 23:208–213.

[Current e Schilling, 1994] Current, J. R. e Schilling, D. A. (1994). The median tour and maximal covering tour problem: Formulations and heuristics. *European Journal of Operational Research*, 73:114–126.

[Dorigo, 1992] Dorigo, M. (1992). *Optmization, Learning and Natural Algorithms*. Tese de Doutorado, Politecnico di Milano.

[Eiben e Rudolph, 1999] Eiben, A. E. e Rudolph, G. (1999). Theory of evolutionary algorithms: A bird's eye view. *Theor. Comput. Sci.*, 229(1):3–9.

[Feo e Resende, 1995] Feo, T. e Resende, M. (1995). Greedy randomized search procedure. *Journal of Global Optimization*, 6:109–133.

[Finke *et al.*, 1984] Finke, G., Claus, A., e Gunn, E. (1984). A two-commodity network flow approach to traveling salesman problem. *Congress. Numerantium*, 41:167–178.

[Gendreau *et al.*, 1992] Gendreau, M., Hertz, A., e Laporte, G. (1992). New insertion and post optimization procedures or the traveling salesman problem. *Operations Research*, 40:1086–1094.

[Gendreau *et al.*, 1995] Gendreau, M., Laporte, G., e Semet, F. (1995). The covering tour problem. *Operational Research*, 45:568–576.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549.

[Glover, 1995] Glover, F. (1995). Scatter search and star paths: Beyond the genetic metaphor. *OR Spektrum*, 17:125–137.

[Glover, 1996] Glover, F. (1996). Tabu search and adaptive memory programming - advances, applications and challenges. *Interfaces in Computer Sciences and Operations Research*. R. Barr and R. Helgason ans J. Kenington, (Eds.).

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Berkeley.

[Hachida *et al.*, 2000] Hachida, M., Hodgson, M. J., e Laporte, G. (2000). Heuristis for the multi-vehicles covering tour problem. *Computers and Operations Research*, 27:29–42.

[Hertz e Kobler, 2000] Hertz, A. e Kobler, D. (2000). A framework for the description of evolutionary algoritms. *European Journal of Operation Research*, 126:1–12.

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gellat, D. C., e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

[Kubik, 2007] Kubik, P. (2007). Heuristic solution approaches for the covering tour problem. Dissertação de Mestrado, Universitat Wien.

[Laporte e Martello, 1990] Laporte, G. e Martello, S. (1990). The selective traveling salesman problem. *Discrete Applied Mathematics*, 26:93–207.

[Liepens e Potter, 1991] Liepens, G. E. e Potter, W. D. (1991). *Handbook of Genetic Algorithms*, chapter A genetic algorithm approach to multi-fault diagnosis, pages 237–250. Van Nostrand Reinhold, New York.

[Lourenço *et al.*, 2003] Lourenço, H. R., Martin, O., e Stützle, T. (2003). *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, Norwell, MA.

[Lyra, 2004] Lyra, A. R. d. (2004). O problema de recobrimento de rotas com coleta de prêmios: Regras de redução, formulação matemática e heurísticas. Dissertação de Mestrado, UFF/IC.

[Maniezzo *et al.*, 1999] Maniezzo, V., Baldacci, R., e Zamboni, M. (1999). *Metaheuristic Optimization via Memory and Evolution*, chapter Scatter Search Methods for the Covering Tour Problem, pages 59–91. Springer US, 2005.

[Mladenovic e Hansen, 1997] Mladenovic, N. e Hansen, P. (1997). Variable neighboorhood search. *Computers and Operations Research*, 24:1097–1100.

[Motta, 2001] Motta, L. C. S. (2001). Novas abordagens para o problema de recobrimento de rotas. Dissertação de Mestrado, UFF/IC.

[Motta *et al.*, 2001a] Motta, L. C. S., Ochi, L. S., e Martinhon, C. A. (2001a). Grasp metaheuristics to the generalized covering tour problem. In *Proceedings of IV Metaheuristic International Conference (IV MIC)*, volume 1, pages 387–393, Porto - Portugal.

[Motta *et al.*, 2001b] Motta, L. C. S., Ochi, L. S., e Martinhon, C. A. (2001b). Reduction rules for the covering tour problems. *Eletronic Notes In Discrete Applied Mathematics*, 7:168–171.

[Motta *et al.*, 2001c] Motta, L. C. S., Ochi, L. S., e Martinhon, C. A. (2001c). Uma metaheurística grasp/vns para uma solução aproximada do problema de recobrimento de rotas. *Série TEMAS - Tendências em Matemática Aplicada e Computacional. SBMAC*, 2:145–154.

[Orman e Williams, 2005] Orman, A. J. e Williams, H. P. (2005). A survey of differente integer programming formulations of the travelling salesman problem.

[Reeves, 2003] Reeves, C. (2003). *Handbook of Metaheuristics*, chapter Genetic Algorithms, pages 55–82. Kluwer Academic Publishers, Norwell, MA.

[Reinelt, 1991] Reinelt, G. (1991). Tsplib - a traveling salesman problem library. *ORSA - Journal on Computing*, pages 376–384.

[Revelle e Laporte, 1993] Revelle, C. S. e Laporte, G. (1993). New directions in plant locations. *Studies in Locational Analysis 5*, pages 31–58.

[Ribeiro *et al.*, 2002] Ribeiro, C. C., Uchoa, E., e Werneck, R. F. (2002). A hybrid grasp with perturbations for the steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246.

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

[Silva *et al.*, 2006] Silva, M. S. A., Mine, M. T., Souza, M. J. F., e Ochi, L. S. (2006). Busca local iterada aplicada ao problema de programação de jogos realizados em dois turnos espelhados. *XXXVIII Simpósio Brasileiro de Pesquisa Operacional. XXXVIII SBPO*, 1:1–8.

[Souza, 2008] Souza, M. J. F. (2008). Inteligência computacional para otimização, notas de aula. http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.pdf.

[Stützle e Hoos, 1999] Stützle, T. e Hoos, H. H. (1999). Analysing the run-time behaviour of iterated local search for the tsp. In *Proceedings of the Third Metaheuristics International Conference*, pages 449–453, Angra dos Reis, Brasil.

[Toth e Vigo, 2002] Toth, P. e Vigo, D. (2002). *The Vehicle Routing Problem*. SIAM.

[Wong, 1980] Wong, R. T. (1980). Integer programming formulations of the traveling salesman problem. In *Proc. IEEE Conf. on Circuits and Computers*, pages 149–152.