DEPARTAMENTO DE COMPUTAÇÃO

D E C O M

Modelagens Exata e Heurística para Resolução do Problema do Caixeiro Viajante com Coleta de Prêmios

U F O P

UNIVERSIDADE FEDERAL DE OURO PRETO

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

Modelagens Exata e Heurística para Resolução do Problema do Caixeiro Viajante com Coleta de Prêmios

Antônio Augusto Chaves

Orientador: Prof. Marcone Jamilson Freitas Souza

OURO PRETO, MG – BRASIL
DEZEMBRO DE 2003

ANTÔNIO AUGUSTO CHAVES

MODELAGENS EXATA E HEURÍSTICA PARA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA DE PRÊMIOS

MONOGRAFIA SUBMETIDA AO DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DE OURO PRETO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Prof. Dr. Marcone Jamilson Freitas Souza
Doutor pela COPPE/Universidade Federal do Rio de Janeiro
Departamento de Ciência da Computação, UFOP

Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti
Doutor pela Universidade Federal de Minas Gerais
Departamento de Ciência da Computação, UFOP

MSc. Roberto Carlos Vieira Pontes
Mestre pelo Instituto Militar de Engenharia, RJ

OURO PRETO, MG – BRASIL
DEZEMBRO DE 2003

Resumo

O Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP), referido na literatura como *Prize Collecting Traveling Salesman Problem*, é uma generalização do Problema do Caixeiro Viajante, podendo ser associado a um caixeiro viajante que coleta um prêmio w_k, não negativo, em cada cidade k que ele visita e paga uma penalidade p_l para cada cidade l que não visita, com um custo c_{ij} de deslocamento entre as cidades i e j. O problema encontra-se em minimizar o somatório dos custos da viagem e penalidades, enquanto inclui na sua rota um número suficiente de cidades que o permitam coletar um prêmio mínimo, w_0, pré-estabelecido. Trata-se de um problema classificado na literatura como NP-difícil, justificando sua abordagem por técnicas heurísticas. O presente trabalho contribui com o desenvolvimento de metaheurísticas híbridas para o PCVCP, baseadas em GRASP e métodos de pesquisa em vizinhança variável (VNS/VND) para solucionar aproximadamente o PCVCP. De forma a validar as soluções obtidas, implementa-se um modelo exato para encontrar a melhor solução para o problema, sendo este modelo aplicado a problemas de pequeno porte. Resultados computacionais demonstram a eficiência da metodologia híbrida proposta, tanto em relação à qualidade da solução final obtida quanto em relação ao tempo de execução.

Palavras-Chaves: Problema do Caixeiro Viajante, Metaheurísticas, GRASP, VNS, VND.

Agradecimentos

À minha família, em especial aos meus avós e aos meus pais, pelo apoio e carinho durante toda a minha vida.

Ao professor Marcone Jamilson Freitas Souza, pela orientação neste trabalho, mas principalmente pela quantidade e qualidade de conhecimento transmitido.

Ao Sr. Roberto Carlos Vieira Pontes pela ajuda na revisão da monografia, contribuindo significativamente para a qualidade desta.

Aos demais professores que de alguma forma contribuíram para a minha formação.

A todos os meus amigos e à República K-Zona pelos maravilhosos quatro anos que aqui passei.

Índice

Lista de Figuras

1 Introdução

1.1 Objetivos do trabalho

Este trabalho tem como objetivo apresentar o Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP), referido na literatura como *Prize Collecting Traveling Salesman Problem* (PCTSP), e propor duas metodologias, uma baseada em programação matemática, dita exata, e outra heurística, para resolvê-lo.

1.2 Importância do trabalho

A escolha deste problema para resolução foi feita em função do fato de que, de nosso conhecimento, até o momento são poucos os trabalhos relacionados a este tema, que é de fácil adaptação a situações da vida real. Em linhas gerais, pode ser descrito como um universo de clientes em potencial, onde existe associado a cada cliente, quando não for atendido, uma penalidade pela expectativa de atendimento ou importância, e quando este for atendido, um ganho relativo. Deseja-se a partir de uma origem, montar um percurso contendo alguns clientes visitados uma única vez retornando ao ponto de partida, minimizando o custo da distância total percorrida e a soma das penalidades, de forma a garantir um ganho mínimo que justifique o investimento.

A dificuldade de solução do PCVCP está no número elevado de soluções existentes. Assumindo que a distância de uma cidade i a outra j seja simétrica, isto é, que $d_{ij} = d_{ji}$, o número total de soluções possíveis é $(n - 1)! / 2$, sendo classificado na literatura como NP-difícil, como apresentado em Melo (2001), isto é, não existem algoritmos que o resolvam em tempo polinomial. Mesmo com os rápidos avanços tecnológicos dos computadores, uma enumeração completa de todas essas soluções é inconcebível para valores elevados de n. Certamente haverá um limite acima do qual tal problema tornar-se-á intratável computacionalmente.

Problemas desta natureza são comumente abordados através de heurísticas. Define-se heurística como sendo uma técnica que procura boas soluções (próximas da otimalidade) a um custo computacional razoável, sem, no entanto, estar capacitada a garantir a otimalidade, bem como garantir quão próxima uma determinada solução está da solução ótima. A grande desvantagem das heurísticas reside na dificuldade de fugir de ótimos locais, o que deu origem à outra metodologia, chamada de Metaheurística, que possui ferramentas que possibilitam sair destes ótimos locais, permitindo a busca em regiões mais promissoras. O grande desafio é produzir, em tempo mínimo, soluções tão próximas quanto possíveis da solução ótima.

Dentre os procedimentos enquadrados como metaheurísticas que surgiram ao longo das últimas décadas, destacam-se: Algoritmos Genéticos (AGs) (Goldberg, 1989), *Simulated Annealing* (Kirkpatrick, 1983), Busca Tabu (BT) (Glover, 1986), *Greedy Randomized Adaptative Search Procedure* (GRASP) (Feo & Resende, 1995), Colônia de Formigas (Taillard, 1999), *Variable Neighborhood Search* (VNS) (Mladenovic & Hansen, 1997), entre outros.

Num problema típico de minimização, encontra-se um ótimo local quando qualquer movimento a ser feito piore o valor atual da função objetivo. Um ótimo global corresponde ao menor valor da função objetivo, entre todos os ótimos locais existentes no espaço de busca.

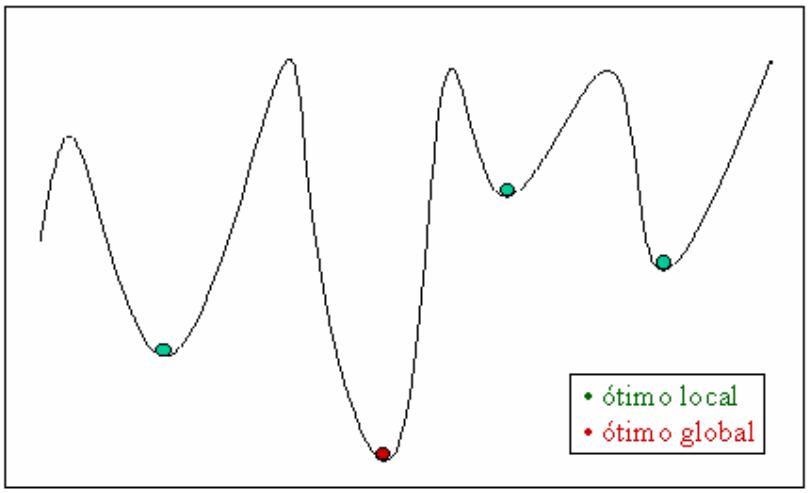

Figura 1.1 – Representação gráfica para um problema de minimização

Para a resolução do PCVCP, faz-se uso de conceitos de técnicas mais recentes, que têm se destacado na solução de problemas altamente combinatórios pelos seus resultados obtidos em diversas aplicações, tais como: GRASP (*Greedy Randomized Adaptive Search Procedure*) (Resende, 1997), *Variable Neighborhood Search* (VNS) (Mladenovic & Hansen, 1997) e *Variable Neighborhood Descent* (VND) (Mladenovic & Hansen, 1997). Há, certamente, outras técnicas que poderiam ser empregadas, mas preferiu-se selecionar GRASP e VNS/VND tendo em vista a simplicidade dessas técnicas e sua eficiência na abordagem de diversos outros problemas combinatórios. Foge ao escopo deste trabalho qualquer comparação entre as metaheurísticas escolhidas e as não utilizadas.

1.3 Estrutura do trabalho

O presente trabalho está dividido em seis seções, incluindo esta introdução.

Na seção II faz-se uma revisão de literatura sobre o PCVCP, e das heurísticas utilzadas para sua resolução neste trabalho.

Na seção III descreve-se o problema do PCVCP, e apresenta-se uma modelagem exata e uma modelagem heurística para o mesmo, e ainda mostra a aplicação dos métodos heurísticos ao problema.

Na seção IV, são mostrados detalhes do sistema desenvolvido.

A seção V mostra os resultados computacionais obtidos pelo sistema.

Finalmente, na seção VI, é feita uma avaliação dos resultados obtidos, além de citar algumas sugestões para trabalhos futuros.

2 Revisão Bibliográfica

2.1 Introdução

As aplicações desenvolvidas para o PCVCP são poucas, apesar da grande aplicabilidade que este tem para o mundo real. Por exemplo, em 1992, Goemans e Willianson desenvolveram um procedimento de 2-aproximativo para uma versão do PCVCP no qual o prêmio mínimo a ser coletado foi removido e o objetivo passou a ser simplesmente minimizar o custo (MELO, 2001).

Em 1994, Dell' Amico et. al. propuseram o uso da relaxação lagrangeana para solucionar a ordem de visitas a clientes de uma empresa num determinado período, cujo objetivo era maximizar o total de valor das solicitações dos clientes menos o custo de deslocamento e venda, estando sujeitos a restrições de tempo. O limite inferior era obtido primeiro removendo as restrições de conectividade, obtendo um problema de associação com restrições de tempo; e depois, através da relaxação lagrangeana, incluindo as restrições de tempo na função objetivo. E em 1995, Awerbuch et. al. desenvolveram o primeiro algoritmo de aproximação para o PCVCP possuindo um desempenho polilogaritmo (MELO, 2001).

Martinhon et. al. (2000) desenvolveram um trabalho que utiliza metaheurísticas híbridas para o PCVCP, no caso, o GRASP associado ao VNS. Nele, também foi investigado o uso de filtragem na fase de construção, ou seja, a utilização de soluções elites, bem como a adoção, no VNS, de estruturas de vizinhança baseadas na troca de nós (inserções e remoções) e troca de arestas.

Recentemente, Melo (2001) também desenvolveu um trabalho combinando as metaheurísticas GRASP e VNS, mas propondo a utilização de uma generalização do GRASP, chamada de GRASP-PSD.

Problemas análogos ao PCVCP encontrados na literatura foram os seguintes:

- PCV Generalizado (PCVG): proposto por Ong (1982), como o próprio nome indica, é uma generalização do PCV, onde dado um conjunto de vértices separados em grupos, deseja-se encontrar a menor rota parcial através de pelo menos um vértice em cada agrupamento, (MELO, 2001).
- PCV Seletivo (PCV-S): proposto por Laporte e Martello (1987), o PCV-S aborda a situação em que, para cada vértice k do grafo, este possui um prêmio w_k, não negativo, o problema consiste na construção de uma rota através de um subconjunto destes vértices, maximizando o total de prêmios coletados pela rota, onde o comprimento não deve exceder a um certo valor R, (GOLDBARG & LUNA, 2000)
- PCV com *Backhauls* (PCVB): esse problema pode ser considerado um caso especial de PCVG onde os nós de G são particionados em dois grupamentos denominados normalmente de L (nós *linehauls*) e B (nós *backhauls*). A matriz de custo do problema atende as condições da norma euclidiana. Uma versão do problema determina que os nós de L sejam visitados inicialmente e, posteriormente, os nós de B. A estratégia dessa versão é dar preferência ao descarregamento dos veículos para, posteriormente, realizar o carregamento em direção ao depósito (nó inicial) (GOLDBARG & LUNA, 2000).

- PCV com Janela de Tempo (PCVJT): dado um conjunto de N vértices, com uma distância d_{ij}, e um tempo de viagem t_{ij} entre os vértices i e j para todos pares de $i,j \in N$, e uma janela de tempo $[a_i, b_i]$ para cada vértice i, onde o vértice i não pode ser visitado antes de a_i e depois de b_i. Se o vértice i for visitado antes de a_i, será necessário esperar um tempo w_i até a_i; mas se o vértice i for visitado depois de b_i, a rota fica inviável. O objetivo é minimizar o custo da rota, onde o custo da rota pode ser a distancia total percorrida (neste caso o tempo de espera é ignorado) ou o tempo total gasto para completar a rota (neste caso o tempo de espera w_i é adicionado ao tempo de viagem t_{ij}) (SIMONETTI, 1998).

2.2 Técnicas Heurísticas

2.2.1 Definição de Heurística

As heurísticas ou algoritmos heurísticos foram desenvolvidos com a finalidade de se resolver problemas de elevado nível de complexidade em tempo computacional razoável. Ao se pensar em um problema altamente combinatório, uma opção seria analisar todas as combinações possíveis para conhecer a melhor. Se o problema possui um universo de dados pequeno, realmente esta é a maneira correta de se buscar a melhor solução, mas os problemas reais, normalmente, possuem um número de combinações muito extenso, o que torna inviável a análise de todas as combinações, uma vez que o tempo computacional exigido fica impraticável. As heurísticas procuram encontrar soluções próximas da otimalidade em um tempo computacional razoável, sem, no entanto, conseguir definir se esta é a solução ótima, nem quão próxima ela está da solução ótima.

Neste trabalho foram utilizadas heurísticas construtivas (*Add* Maior Economia e *Drop* Maior Economia) que têm a finalidade de construir uma solução inicial buscando minimizar a função objetivo, e heurísticas de refinamento (*k-Optimal*, *AddDrop*, Seq*Drop*Seq*Add*) que são utilizadas para refinar uma solução, ou seja, tentar encontrar uma solução de melhor qualidade a partir de modificações efetuadas em outra solução.

2.2.2 Definição de Metaheurísticas

O exemplo apresentado a seguir foi apresentado em Melo (2001), onde se faz uma analogia de metaheurística com um problema real.

Imagine a seguinte situação: um grupo de pesquisadores, em atividade numa floresta tropical, ficou perdido. Como a carga da bateria do único celular da expedição, estava acabando, somente conseguiram avisar à base da situação atual e que estavam presos no vale mais profundo de toda região. De posse dessas informações, foi criado um grupo de resgate. Levando em consideração que não existia estudo topográfico sobre a região, que era extensa, o grupo de resgate viu-se dividido entre três opiniões distintas: A primeira sugestão dada foi a de que um avião tentasse percorrê-la na íntegra, identificando todos os vales ali contidos para que ao término fossem comparados e o menor deles seria o local exato para o resgate. A segunda sugestão, era a de que a cada dia fosse escolhida aleatoriamente uma direção, e nela fossem também identificados os vales existentes e ao término de alguns dias, seria escolhido o vale mais profundo até então, e se tentaria o resgate. A terceira e última sugestão era o meio termo entre as duas primeiras. Baseado nas informações colhidas do grupo de pesquisadores

nos dias anteriores, a idéia seria utilizá-las para que se determinasse conjunto de regiões menores e somente ali fosse intensificada a procura pelo vale mais profundo.

Estas três idéias possuem os seguintes enfoques:

- A primeira idéia, que certamente acharia o local exato para o resgate, conhecida em otimização como métodos exatos, não seria aceita, porque o tempo gasto na procura comprometeria a integridade física dos pesquisadores. Analogamente, nos métodos exatos, busca-se encontrar, a partir de uma região de busca *X*, a solução ótima *x* analisando cada elemento desta região.
- A segunda idéia, provavelmente também não obteria êxito nas buscas, pois estas seriam realizadas sem qualquer informação prévia da região a ser pesquisada. A esta idéia, atribuímos as chamadas heurísticas cegas, cuja excessiva flexibilidade podem conduzir a busca a resultados caóticos.
- A terceira idéia, conjuga aspectos que permitem se esquivar dos erros ocorridos nas idéias anteriores, pois leva em consideração todas as informações previamente conhecidas do espaço de busca a ser explorado, o que em otimização podem ser descritos como metaheurísticas.

Problemas de otimização existem nas mais diversas áreas de aplicação, como telecomunicação, planejamentos operacionais na fabricação de semicondutores, desenhos de áreas escolares, localização de reservas energéticas estratégicas, roteamento de veículos, organização de tropas, planejamento de tripulação de aeronaves, alocação de salas de aula em escolas, escalonamento de jogos, entre outros. Teoricamente é possível enumerar todas as soluções e avaliar cada uma a respeito do objetivo esperado, encontrando a solução ótima. Porém, vários estudos e também este trabalho demonstram que seguir esta estratégia é tecnicamente inviável, pois o número de combinações freqüentemente cresce exponencialmente com o tamanho do problema.

Muitos trabalhos foram desenvolvidos nas últimas décadas com o sentido de melhorar os métodos heurísticos, sem no entanto prejudicar a sua principal característica, que é a flexibilidade. Estes trabalhos deram origem as estratégias comumente conhecidas como metaheurísticas.

Metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. A principal característica das metaheurísticas é a capacidade que estas possuem de escapar de ótimos locais.

Neste trabalho foram utilizadas as metaheurísticas GRASP, VNS e VND, descritas a seguir.

2.2.3 *Add* Maior Economia

Uma solução inicial de boa qualidade é muito importante, uma vez que bons pontos de partida permitem acelerar a busca local. Este método proposto por Melo (2001) procura construir uma solução com qualidade.

Procura-se compor uma rota parcial, garantindo que o somatório dos prêmios coletados seja maior ou igual ao prêmio mínimo exigido, para tal, baseia-se no Método das Economias, conhecido na literatura como *Savings*, originalmente proposto por Clarke e

Wright para o Problema de Roteamento de Veículos, e no método *Generalized Savings*, proposto por Golden *et. al.* (MELO, 2001).

Inicialmente tem-se uma rota R partindo e retornando à origem. O valor da função objetivo é igual ao somatório de todas as penalidades. O valor do prêmio coletado é igual a zero, pois o prêmio para a origem corresponde a zero.

Definida a rota inicial, para cada vértice k não pertencente à rota R, verifica-se sua economia atual em relação a todas as arestas (i, j) que formam a rota. A função do cálculo para cada inserção de k é então definida da seguinte forma:

$$economia\ (k) \ = \ g_k \ = \ \max\ (i,j)\ \{\ c_{ij} + p_k - c_{ik} - c_{kj}\ \},\ \forall\ k \notin R \qquad (2.1)$$

sendo composta pelo custo da aresta (i,j), a penalidade do vértice k e os custos das arestas (i,k) e (k,j), respectivamente.

Após isto, seleciona-se dentre todos os vértices não pertencentes à rota parcial atual o que apresentar a maior economia positiva e este é inserido em R. Feito isto, para cada vértice k não pertencente à rota R, faz-se o recálculo da economia em relação a todas as arestas (i,j) que pertencem a rota R. Note, que se $g_k > 0$, após a inserção do vértice k, obtêm-se uma redução no custo da função objetivo.

O método termina quando o somatório de todos os prêmios dos vértices pertencentes a R for igual ou maior que o prêmio mínimo e não haja mais vértice k com economia positiva.

Pode-se descrever um algoritmo básico de inserção *Add* Maior Economia, conforme o descrito na figura 2.1, a seguir:

```
Procedimento Add_Maior_Economia
Inicialização
  R ← Ø;
  Inserir a origem em R;
  Para todo k não pertencente a R faça
     Calcular a economia de inserção;
  Fim Para
  Enquanto não for atingido o prêmio mínimo ou
                existir alguma economia positiva faça
     k ← Vértice de maior economia para inserção;
     Inserir o Vértice k em R;
     Atualizar valores;
  Fim Enquanto
Fim Procedimento
```

Figura 2.1 – Algoritmo de inserção *Add* Maior Economia

2.2.4 *Drop* Maior Economia

Este método, proposto por Melo (2001), parte de uma solução que contém todos os vértices e utiliza uma abordagem oposta à dos algoritmos de inserção, isto é, a cada iteração retira-se um vértice segundo o cálculo de uma função gulosa.

Definida uma solução inicial, com uma rota R que contenha todos os vértices, para cada vértice k pertencente à rota, calcula-se a economia associada à remoção deste vértice. A função do cálculo para remoção do vértice k é definida por:

$$economia\ (k) \ = \ g_k = max\{\ c_{ak\ ,k} + c_{k\ ,sk} - c_{ak,sk} - p_k\ \},\ \forall\ k \in R \qquad (2.2)$$

onde a_k e s_k representam, respectivamente, o antecessor e o sucessor do vértice k na rota R.

Após isto, seleciona-se dentre todos os vértices pertencentes à rota atual o que apresentar a maior economia positiva e este é removido de R. Feito isto, para cada vértice k pertencente à rota R, faz-se um recálculo da economia para remoção. Note que se $g_k > 0$, a remoção do vértice k proporcionará uma redução no custo da função objetivo.

O método continua enquanto existir algum vértice com economia positiva e a retirada deste vértice não acarretar uma solução inviável, ou seja, o somatório dos prêmios dos vértices pertencentes à rota for menor que o prêmio mínimo pré-estabelecido.

O método *Drop* Maior Economia pode ser descrito conforme a figura 2.2, a seguir.

```
Procedimento Drop_Maior_Economia
Continua ← verdadeiro;
Inicialização
  Obter a rota R com todos os vértices;
  Para todo k pertencente a R faça
    Calcular a economia de remoção;
  Fim Para
  Enquanto Continua faça
    k ← Vértice de maior economia para remoção;
    Se (g_k > 0) e (W - w_k > wo) então
        Remover o Vértice k de R;
        Atualizar valores;
    Senão
        Continua ← falso;
    Fim Se   Fim Enquanto
Fim Procedimento
```

Figura 2.2 – Algoritmo de remoção *Drop* Maior Economia

onde W é o somatório dos prêmios coletados na rota, w_k é o prêmio do vértice k e w_0 é prêmio mínimo que precisa ser coletado para que a rota não seja inviável.

2.2.5 Método de Descida k-*Optimal*

Após realizar a fase de construção, tem-se uma rota com alguns (ou todos) vértices, e sabe-se qual a seqüência em que estes vértices são visitados. O desejo agora é conseguir uma possível melhora na função objetivo, através da tentativa de mudança na sua ordem de visitas, o que pode ser conseguido realizando-se possíveis trocas das suas arestas.

As heurísticas de substituição, Costa Jr. (2003), são estratégias de melhoria. Partindo de um ciclo hamiltoniano H, excluem-se k arestas de H, produzindo k caminhos desconectados. Reconectam-se estes k caminhos de alguma maneira para produzir outro ciclo H', usando diferentes arestas daquelas que foram removidas de H. Desta maneira, H e H' serão diferentes entre si por exatamente k arestas. São verificadas todas as soluções viáveis contendo H', escolhendo-se a melhor dentre todas, chamada de solução *k-Opt*.

À medida que o valor de k aumenta, em geral, aumenta também a probabilidade de se obter à solução ótima. Entretanto, o custo computacional também cresce rapidamente com o

valor de *k*, pois a complexidade deste algoritmo é O(n^k) (GOLDBARG & LUNA, 2000). Neste trabalho optou-se em utilizar a heurística 2-*Optimal*, por esta ser eficiente e computacionalmente barata.

Um algoritmo *k-Optimal* pode ser descrito conforme a figura 2.3, a seguir.

```
Procedimento k-Optimal
    Seja s_0 uma solução inicial;
    s ← s_0;
    s* ← s;
    Repetir
        Q ← Lista de todos subconjuntos de k arestas de R;
        Enquanto Q > Ø  faça
            Z ← Lista com k arestas de Q;
            Remover as arestas Z de R gerando R';
            Construir todas possíveis rotas que contenham R';
            s* ← s que apresentar maior melhoria;
            Q ← Q - Z;
        Fim enquanto
        Se f(s*) < f(s) então
            s ← s*;
        Fim Se
    Até f(s) ≥ f(s*)
    Retornar s;
Fim Procedimento
```

Figura 2.3 – Algoritmo k-*Optimal*

2.2.6 Método *AddDrop*

Este algoritmo baseia-se nos critérios de inserção e remoção de vértices vistos anteriormente. Funcionando de maneira iterativa, através de sucessivas tentativas de troca da solução corrente por uma solução melhor através de uma inserção seguida de uma remoção, conforme mostrado abaixo:

- *Add* – ocorrerá sempre uma inserção, independente de existir melhora ou não no valor da função objetivo;
- *Drop* – ocorrerá sempre uma remoção, idependendete de existir melhora ou não no valor da função objetivo.

É importante ressaltar que não será permitido inviabilizar a solução, isto é, permitir que o prêmio coletado seja menor que o prêmio mínimo pré-estabelecido. As inserções ocorrerão segundo o método *Add* Maior Economia, e as remoções segundo o método *Drop* Maior Economia.

2.2.7 Método de Descida SeqDropSeqAdd

Este algoritmo também se baseia nos critérios de remoção e inserção de vértices vistos anteriomente. Funcionando de maneira iterativa, ou seja, através de sucessivas tentativas de

troca da solução corrente por uma solução melhor na sua vizinhança, o que é conseguido através de *n* remoções seguida de *m* inserções, conforme mostrado abaixo:

- SeqDrop – as remoções ocorrerão enquanto houver redução no valor da função objetivo;
- SeqAdd – as inserções ocorrerão enquanto houver redução no valor da função objetivo.

Não permitindo que o prêmio mínimo coletado seja menor que o prêmio mínimo pré-estabelecido. As remoções ocorrerão segundo o método Drop Maior Economia e as inserções segundo o método Add Maior Economia.

2.2.8 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*), Resende (1997), pode ser visto como uma metaheurística que utiliza as boas características dos algoritmos puramente gulosos e dos procedimentos aleatórios na fase de construção de soluções viáveis.

O GRASP é um processo iterativo, no qual cada iteração consiste em duas fases distintas: a fase de construção, onde uma solução viável é construída, e a fase de busca local, onde um ótimo local na vizinhança da solução inicial construída é encontrado. A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado. Um algoritmo GRASP pode ser descrito conforme a figura 2.4 a seguir.

```
Procedimento GRASP
   f(s) ← ∞;
   Para k de 1 até MaxIter faça
      Aplicar o procedimento de construção para obter
       uma solução viável s ;
      Aplicar busca local em s gerando uma nova
       solução s' ;
      Se f (s') < f (s) então
          s = s' ;
   Fim Para
   Retornar s ;
Fim Procedimento
```

Figura 2.4 – Algoritmo GRASP

Na fase de construção, uma solução é iterativamente construída, elemento por elemento. A cada iteração dessa fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista *C* de candidatos, seguindo um critério de ordenação pré-determinado. Esse processo de seleção é baseado em uma função adaptativa gulosa $f: C \rightarrow R$, que estima o benefício da seleção de cada um dos elementos. A heurística é adaptativa porque os benefícios associados com a escolha de cada elemento são atualizados em cada iteração da fase de construção para refletir as mudanças oriundas da seleção do elemento anterior. O componente probabilístico do procedimento reside no fato de que cada elemento é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos restrita (LCR). Esta técnica de escolha permite

que diferentes soluções sejam geradas em cada iteração GRASP. Um parâmetro $\alpha \in [0,1]$ controla o nível de gulosidade e aleatoriedade da fase de construção. Um valor $\alpha = 0$ faz gerar soluções puramente gulosas, enquanto que $\alpha = 1$ faz produzir soluções totalmente aleatórias. Um algoritmo da fase de construção do GRASP pode ser descrito conforme a figura a seguir.

```
Procedimento Construção GRASP
   s ← Ø ;
    Inicializar o conjunto de candidatos C;
    Enquanto C ≠ Ø faça
       t_I = min {g (t) | t ∈ C} ;
       t_S = max {g (t) | t ∈ C} ;
       RLC={ t ∈ C | g (t) < t_I + α (t_S - t_I) };
       Selecione aleatoriamente, um elemento t ∈ RLC ;
       s ← s ∪{s};
       Atualizar o conjunto de candidatos C;
    Fim Enquanto
    Retornar s;
Fim Procedimento
```

Figura 2.5 – Fase de Construção de um algoritmo GRASP

Assim como em muitas técnicas determinísticas, as soluções geradas pela fase de construção do GRASP provavelmente não são localmente ótimas com respeito à definição de vizinhança adotada. Daí a importância da fase de busca local, a qual objetiva melhorar a solução construída. Um algoritmo de busca local pode ser descrito conforme a figura a seguir.

```
Procedimento Busca Local GRASP
    s ← solução inicial;
    Enquanto critério de parada não satisfeito faça
       Encontrar uma solução s' ∈ N(s);
       Se f(s') < f(s) então
       s ← s';
    Fim Enquanto
    Retornar s;
Fim Procedimento
```

Figura 2.6 – Algoritmo básico de busca local GRASP

A eficiência da busca local depende da qualidade da solução construída. A fase de construção tem então um papel importante na busca local, uma vez que se as soluções construídas constituem bons pontos de partida para a busca local, permitindo assim acelerá-la.

O parâmetro α, que determina o tamanho da lista de candidatos restrita, é basicamente o único parâmetro a ser ajustado na implementação de um procedimento GRASP. Sabe-se que valores de α que levam a uma LCR de tamanho muito limitado (ou seja, valor de α próximo da escolha gulosa) implicam soluções finais de qualidade muito próxima àquela obtida de forma puramente gulosa, obtidas com um baixo esforço computacional. Em contrapartida, provocam uma baixa diversidade de soluções construídas. Já uma escolha de α próxima da

seleção puramente aleatória leva a uma grande diversidade de soluções construídas, mas por outro lado, muitas das soluções construídas são de qualidade inferior, tornando mais lento o processo de busca local.

O procedimento GRASP procura, portanto, conjugar bons aspectos dos algoritmos puramente gulosos com aqueles dos procedimentos aleatórios de construção de solução.

2.2.9 VNS

O Método de Pesquisa em Vizinhança Variável (*Variable Neighborhood Search*, VNS), Mladenovic & Hansen (1997), é um método de busca local que consiste em explorar o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança. Contrariamente a outras metaheurísticas baseadas em métodos de busca local, o método VNS não segue uma trajetória, mas sim explora vizinhanças gradativamente mais "distantes" da solução corrente e focaliza a busca em torno de uma nova solução, se e somente se, um movimento de melhora é realizado. O método inclui, também, um procedimento de busca local a ser aplicado sobre a solução corrente. Esta rotina de busca local também pode usar diferentes estruturas de vizinhança. Um algoritmo de VNS pode ser descrito conforme a figura a seguir.

```
Procedimento VNS
    Seja s_0 uma solução inicial;
    Seja r o número de estruturas diferentes de vizinhança;
    s ← s_0 ;
    Enquanto critério de parada não satisfeito faça
        k ← 1
        Enquanto k ≤ r faça
            Gere um vizinho qualquer s' ∈ N(k)(s)
            s'' ← Busca Local (s')
            Se f(s'') < f(s) então
                s ← s'' ;
                k ← 1;
            Senão
                k ← k + 1;
        Fim Enquanto
    Fim Enquanto
    Retorne s
Fim Procedimento;
```

Figura 2.7 – Algoritmo VNS

Nesse algoritmo, parte-se de uma solução inicial qualquer e a cada iteração seleciona-se aleatoriamente um vizinho s' dentro da vizinhança $N^{(k)}(s)$ da solução s corrente. Esse vizinho é então submetido a um procedimento de busca local. Se a solução ótima local, s'', for melhor que a solução s corrente, a busca continua de s'' recomeçando da primeira estrutura de vizinhança $N^{(1)}(s)$. Caso contrário, continua-se a busca a partir da próxima estrutura de vizinhança $N^{(k+1)}(s)$. Este procedimento é encerrado quando uma condição de parada for atingida, tal como o tempo máximo de iterações consecutivas entre dois melhoramentos. A solução s' é gerada aleatoriamente, de forma a evitar ciclagem, situação que pode ocorrer se alguma regra determinística for usada.

2.2.10 VND

O Método de Descida em Vizinhança Variável (Variable Neighborhood Descent, VND), Mladenovic e Hansen (1997), também é um método de busca local que consiste em explorar o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança, aceitando somente soluções de melhora da solução corrente e retornando a primeira estrutura quando uma solução melhor é encontrada. Todas as vizinhanças são verificadas, e depois de analisada a última, ocorre o critério de parada. Este método pode ser descrito conforme a figura a seguir:

```
Procedimento VND
    Seja s_0 uma solução inicial;
    Seja r o número de estruturas diferentes de vizinhança;
    s ← s_0 ;
    k ← 1
    Enquanto k ≤ r faça
        Encontre o melhor vizinho s' ∈ N(k)(s) ;
        Se f(s') < f(s) então
            s ← s' ;
            k ← 1;
        Senão
            k ← k + 1;
    Fim Enquanto
    Retorne s;
Fim Procedimento;
```

Figura 2.8 – Algoritmo VND

3 Metodologia

3.1 Descrição do problema

O PCVCP foi formulado inicialmente em 1985 por Egon Balas, (Balas, 2001), como um modelo para a programação da operação diária de uma fábrica que produzia lâminas de aço. Por razões que tinham a ver com o desgaste dos rolos e também por outros fatores, a seqüência na ordem do processamento era essencial. A programação consistia na escolha de um número de lâminas associadas às suas ordens de execução, que satisfizessem o limite inferior do peso, e que ordenadas numa seqüência apropriada, minimizasse a função de seqüência. As tarefas de escolha das lâminas e das opções disponíveis para seu o sequenciamento, necessitavam ser resolvidas em conjunto. Chamado então de *Prize Collecting Traveling Salesman Problem*, ou seja, Problema do Caixeiro Viajante com Coleta de Prêmios, serviu como base para o desenvolvimento de um software implementado por Balas e Martin em 1986, (Balas, 2001), que utilizava a combinação de várias heurísticas para encontrar soluções próximas do ótimo local, organizando-as em programações diárias.

O problema do caixeiro viajante (PCV), Stützle & Dorigo (1999), é um dos mais tradicionais e conhecidos problemas de programação matemática. Os problemas de roteamento lidam em sua maior parte com rotas sobre pontos de demanda ou oferta. Esses pontos podem ser representados por cidades, postos de trabalho ou atendimento, clientes, depósitos etc. O PCV é descrito por um conjunto de n cidades e uma matriz de distância entre elas, tendo o seguinte objetivo: o caixeiro viajante deve sair de uma cidade chamada origem, visitar cada uma das n - 1 cidades restantes apenas uma única vez e retornar à cidade origem percorrendo a menor distância possível, ou seja, deve ser encontrada uma rota fechada (ciclo hamiltoniano) de comprimento mínimo que passe exatamente uma única vez por cada cidade.

Um grande número de problemas de roteamento e planejamento pode ser formulado como uma generalização do Problema do Caixeiro Viajante. Neste caso, o PCVCP pode ser associado a um caixeiro viajante que coleta um prêmio w_k, não negativo, em cada cidade k que ele visita e paga uma penalidade p_l para cada cidade l que não visita, com um custo c_{ij} de deslocamento entre as cidades i e j. O problema encontra-se em minimizar o somatório dos custos da viagem e penalidades, enquanto inclui na sua rota um número suficiente de cidades que permitam coletar um prêmio mínimo, w_0, pré-estabelecido.

Caso o valor de w_0 seja igual ao somatório de todos os prêmios w_k, para cada cidade k, tem-se o Problema do Caixeiro Viajante.

O PCVCP pode ser representado conforme a figura a seguir, onde se tem um depósito e alguns vértices formando um grafo completo, ou seja, todos os vértices estão ligados por uma aresta, e existe um custo de deslocamento em cada aresta. Para cada vértice i existe também um prêmio (w_i) e uma penalidade (p_i), ressaltando que para o depósito o prêmio é zero e a penalidade é infinita.

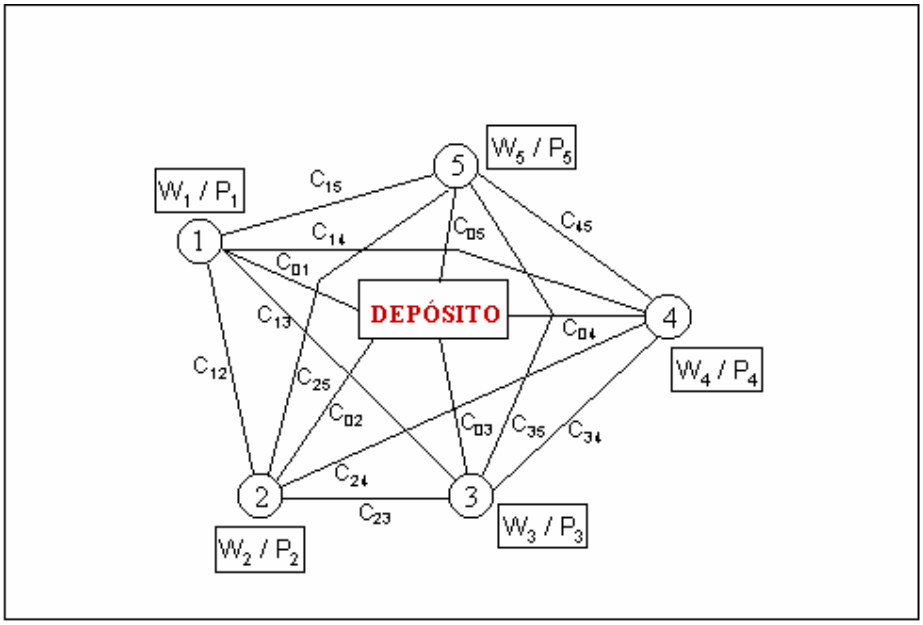

Figura 3.1 – Uma representação do PCVCP

A partir do grafo apresentado na figura 3.1, uma possível solução para o PCVCP é demonstrada a seguir. Constrói-se uma sub-rota a partir do depósito objetivando minimizar o custo e a penalidade, além de coletar no mínimo um prêmio *wo*. Uma solução seria a sub-rota R_1 = {0,4,5,2,3,0}, isto é, o caixeiro sai do depósito (0) com um prêmio igual 0, constrói a sua rota passando pelas cidades 4,5,2 e 3, coletando um prêmio igual a W ($w_4 + w_5 + w_2 + w_3$), e retornando ao depósito, o caixeiro tem um custo de viajem igual a C ($c_{04} + c_{45} + c_{52} + c_{23} + c_{30}$), este também paga uma penalidade igual a P (p_1), por não ter visitado a cidade 1.

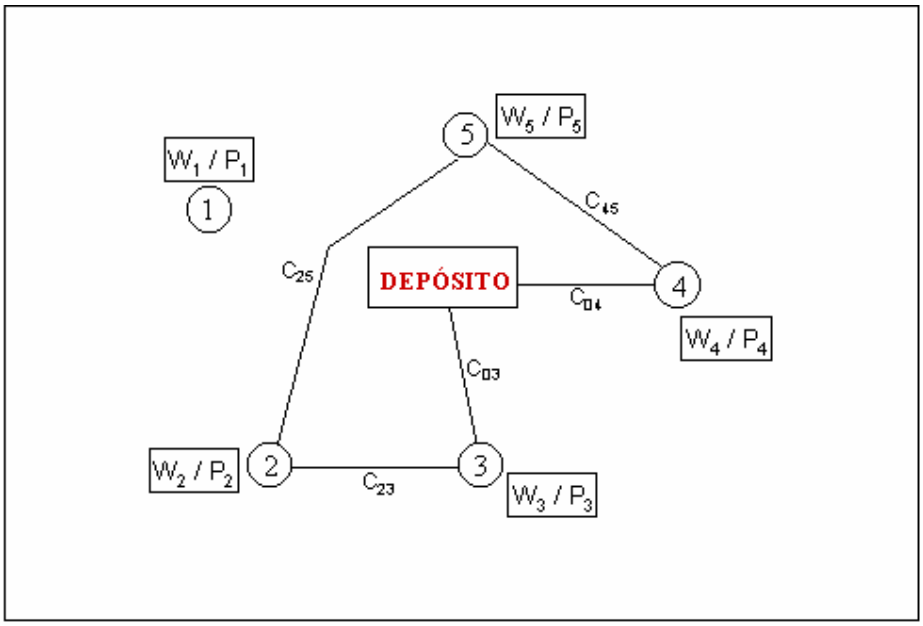

Figura 3.2 – Uma possível solução para o PCVCP

3.2 Modelagem Exata

3.2.1 Introdução

A modelagem exata visa encontrar a melhor solução para o problema, ou seja, o ótimo global, mas devido ao número elevado de soluções existentes para o PCVCP em dimensões elevadas, torna-se inconcebível uma enumeração completa de todas. Mas apesar disto esta modelagem é de fundamental importância neste trabalho, pois servirá para validar o modelo heurístico.

3.2.2 Formulação Matemática

Seja $G' = (N, A)$ um grafo completo direcionado, para cada arco (i,j) de A é dado um custo c_{ij}, e para cada vértice i de N, é associada uma penalidade p_i, a ser paga se o vértice i não compor a rota. Adicionalmente, para cada vértice i, existe um prêmio w_i associado. Os vértices são numerados de 0 até $n = |N|$, sendo o vértice 0, sem perda de generalidade, assumido como depósito ou domicílio do caixeiro viajante. Para este vértice especial, será utilizado $w_0 = 0$ e $p_0 = \infty$. Segue uma formulação proposta em 1985 por Egon Balas, Balas (2001), na qual as restrições 3.5 e 3.6 são propostas neste trabalho para eliminar a existência de subrotas.

Assumindo que y_i seja 1 se o vértice i for incluído na rota e 0 caso contrário, que x é o vetor de incidência associado à rota (ou seja, assume valor 1 caso a aresta i,j esteja na rota, e 0 caso contrário), e que f garante que a diferença entre o fluxo que chega ao vértice e que sai do vértice seja igual a 1, se o vértice for visitado, e 0 caso contrário, e também não permite que a quantidade de fluxo entre os vértices i e j ultrapasse o número de vértices possíveis de serem visitados, então o Problema do Caixeiro Viajante com Coleta de Prêmios pode ser formulado como:

$$\text{(PCVCP) minimizar} \quad \sum_{i \in N} \sum_{j \in N-\{1\}} c_{ij} x_{ij} + \sum_{i \in N} p_i (1 - y_i) \tag{3.1}$$

sujeito à :

$$\sum_{j \in N-\{i\}} x_{ij} = y_i \qquad \forall\, i = 1,\ldots,n \tag{3.2}$$

$$\sum_{i \in N-\{j\}} x_{ij} = y_j \qquad \forall\, j = 1,\ldots,n \tag{3.3}$$

$$\sum_{i \in N} w_i * y_i \geq wmin \tag{3.4}$$

$$\sum_{i \in N} \sum_{j \in N} f_{ij} - \sum_{i \in N} \sum_{j \in N} f_{ji} = y_i \qquad \forall\, i = 1,\ldots,n \;\; i \neq 1 \tag{3.5}$$

$$f_{ij} \leq (n-1) x_{ij} \qquad \forall\, (i,j) \in A \tag{3.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{3.7}$$

$$y_j \in \{0,1\} \qquad \forall\, i = 1,\ldots,n \tag{3.8}$$

A restrição 3.2 diz que se o vertice i estiver na rota, o somatório das arestas que saem dele tem que ser igual a 1, e 0 caso contrário, a restrição 3.3 diz que se o vértice j estiver na rota, o

somatório das arestas que chegam nele tem que ser igual a 1, e 0 caso contrário. A restrição 3.4 assegura que o prêmio coletado na rota será maior ou igual ao prêmio mínimo pré-estabelecido. As restrições 3.5 e 3.6 foram propostas neste trabalho para eliminar sub-rotas, a restição 3.5 garante que o fluxo que chega e que sai do vértice *i* seja igual a 1 caso o vértice seja visitado, e 0 caso contrario, e a restição 3.6 assegura que o fluxo máximo que pode passar por uma aresta é o número de vértices menos 1, ou seja, (*n*-1), pois como tenho *n* vértices o número máximo de arestas possíveis em uma solução é *n* - 1. As restrições 3.7 e 3.8 asseguram a integralidade e não negatividade das variáveis *x* e *y*, respectivamente.

3.2.3 Implementação e Validação do modelo

O modelo matemático proposto na seção anterior pode ser classificado como um modelo de Programação Linear (PL), uma vez que, tanto a função objetivo como as restrições são equações/inequações lineares. Como as variáveis do modelo somente admitem valores inteiros então este é um modelo de Programação Linear Inteira, ou simplesmente de Programação Inteira (PRADO, 1999).

A PL é uma técnica de otimização utilizada para encontrar o ótimo global, seja ele máximo ou mínimo, em situações nas quais temos diversas alternativas de escolha sujeitas a algum tipo de restrição ou regulamentação.

Para a resolução deste modelo de Programação Linear Inteira, fez-se uso do *software* LINGO versão 7.0, objetivando encontrar a solução ótima para o PCVCP. Observa-se, entretanto, pela natureza combinatorial do problema abordado, que tal modelo só consegue resolver problemas de pequenas dimensões.

A entrada e saída de dados utilizadas neste trabalho para a resolução do PCVCP serão mostradas nas figuras a seguir.

Matriz de Custos

Cidades	0	1	...	*n*
0	0	c_{01}	...	c_{0n}
1	c_{10}	0	...	c_{1n}
...	...	...	...	...
n	c_{n1}	c_{n2}	...	0

Cidades	0	1	...	*n*
Prêmio	0	w_1	...	w_n

Cidades	0	1	...	*n*
Penalidade	10000000	p_1	...	p_n

Premio Mínimo	wmin

Figura 3.3 – Entrada de dados para a resolução exata do PCVCP

Na figura 3.3, temos uma tabela de custos simétrica com os valores dos custos de deslocamento entre as cidades *i* e *j*, sendo que quando a cidade *i* for igual à cidade *j* o custo de deslocamento será 0, uma vez que não se pode criar um *loop* em nenhuma cidade. Temos também as tabelas de prêmios e penalidades com os respectivos prêmios e penalidade para cada vértice, e o prêmio mínimo pré-estabalecido.

Matriz Solução

Cidades	0	1	...	*n*
0	0	1	...	0
1	0	0	...	1
...	...	...	...	...
n	1	0	...	0

Figura 3.4 – Saída de dados da resolução exata do PCVCP

Na figura 3.4 temos a tabela utilizada para guardar a rota obtida como resultado do PCVCP, onde se a aresta (*i*, *j*) for visitada terá valor 1 e caso contrário terá valor 0.

A implementação deste modelo matemático pode ser descrita conforme a figura 3.5.

```
model:
title PCVCP;
sets:
       cidades/@ole('dados.xls','cidade') / : p, y, w;
       matriz(cidades,cidades) : x, c, f;
endsets
data:
        c = @ole('dados.xls', 'custo');
        p = @ole('dados.xls', 'penalidade');
       w = @ole('dados.xls', 'premio');
       wo = @ole('dados.xls', 'wo');
enddata

min = fo = @sum( matriz(i,j) | j #ne# i : c(i,j) * x(i,j) ) +
              @sum( cidades(i) : p(i) * (1-y(i) ) );

@for( cidades(i) : @sum( cidades(j) | j #ne# i: x(i,j) ) = y(i) );
@for( cidades(j) : @sum( cidades(i) | i #ne# j: x(i,j) ) = y(j) );
@sum( cidades(i) : w(i)*y(i) )  >=  wo;
@for( matriz(i,j) : @bin( x(i,j) ) );
@for( cidades(i) : @bin( y(i) );  x(i,i) = 0; );
@for( cidades(i) | i #ne# 1:
          @sum( cidades(j) : f(i,j) )  -  @sum( cidades(j) : f(j,i) ) = y(i) );
n = @size( cidades );
@for( matriz(i,j) : f(i,j) <= (n-1) * x(i,j) );

data:
      @ole('dados.xls','solucao','fo')  =  x, fo;
enddata
```

Figura 3.5 – Implementação do Modelo Matemático do PCVCP

A validação do modelo se deu através de vários testes, onde se utilizou grafos completos e simétricos, com diferentes quantidades de vértices.

Através destes testes e também da literatura existente sobre o assunto, verificou-se que a modelagem exata do PCVCP se torna inviável à medida que o número de cidades (vértices) aumenta, pois se torna impossível enumerar todas as possibilidades, uma vez que o número de combinações cresce exponencialmente com o tamanho do problema.

A figura a seguir ilustra os testes realizados, percebendo-se que para um número de cidades próximo a 50, a obtenção da solução por esta metodologia exata já se torna inviável computacionalmente. Este fato demonstra a necessidade de se implementar modelos heurísticos para a resolução do PCVCP de dimensões mais elevadas. Tais modelos, apesar de não garantirem a otimalidade da solução final, têm capacidade de encontrar boas soluções a um custo computacional razoável e podem ser validados através do método exato desenvolvido neste trabalho.

Execução da Modelagem Exata

Instância	| V |	Nº de iterações	Tempo	Ótimo Global
v10	11	1896	00:01	1765
v20	21	173375	01:05	2302
v30a	31	207408	01:26	3582
v30b	31	289466	01:40	2515
v30c	31	5303358	29:46	3236
v50a	51	284015587	180:00	Não encontrado
v50b	51	305687245	180:00	Não encontrado

Figura 3.6 – Resultados encontrados com a modelagem exata

A figura 3.6 ilustra o número de cidades que o problema possui (|V|), o número de iterações que o LINGO executou até encontrar o ótimo global, o tempo (em minutos) gasto para isto, e se o ótimo global para o PCVCP foi encontrado, considerando os valores dos dados utilizados para testes. Destaca-se que, para 50 cidades, não só não se encontrou o ótimo global, como também não se encontrou nenhuma solução viável em 3 horas de execução.

3.3 Modelagem Heurística

3.3.1 Introdução

Como apresentado na seção anterior, o modelo exato não consegue resolver em tempo computacionalmente viável o problema do PCVCP em dimensões elevadas, sendo necessária então a implementação de um modelo heurístico, sendo que este procura encontrar boas soluções a um custo computacional razoável. O grande desafio desta modelagem é produzir, em tempo mínimo, soluções tão próximas quanto possíveis da solução ótima. Como já foi dito anteriormente, para se resolver o PCVCP fez-se uso de metaheurísticas híbridas, baseadas nas metodologias GRASP, VNS e VND.

3.3.2 Representação

A estrutura de entrada dos dados utilizada na implementação do modelo heurístico é praticamente a mesma utilizada no modelo exato. Mais precisamente, tem-se uma matriz de custos entre os vértices (representando as distâncias entre as cidades), e dois vetores com o prêmio e a penalidade de cada vértice. Nesta representação, o vértice *0* denota o depósito, possuindo prêmio nulo e penalidade infinita. A figura 3.7 ilustra esta estrutura de dados.

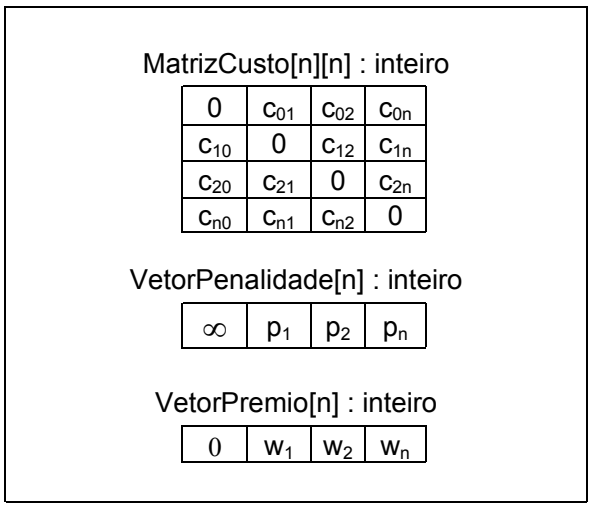

Figura 3.7 – Modelagem da estrutura de entrada dos dados

Os vértices que compõem o grafo do problema são armazenados em uma lista, chamada Candidatos, e para se construir uma solução, a cada etapa é removido um vértice desta lista e inserido em outra lista, chamada de Solução. A figura a seguir ilustra estas duas listas.

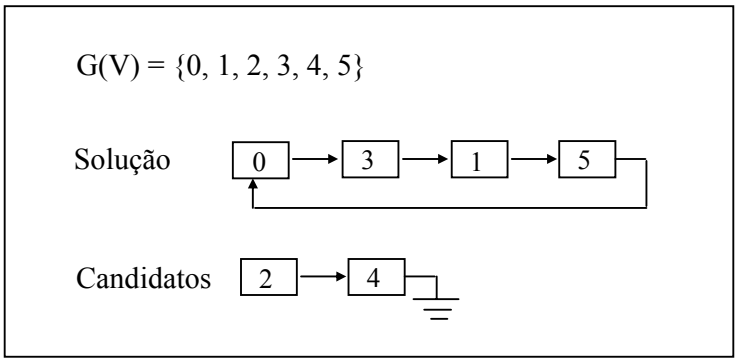

Figura 3.8 – Modelagem da solução

3.3.3 Tipos de Movimentos

Para resolução do PCVCP implementou-se cinco tipos de movimentos, os quais serão definidos a seguir:

m_1: Retirar vértice de maior economia: Utilizando-se o método *Drop* Maior Economia, escolhe-se o vértice que compõe a rota e possui a maior economia positiva e este é retirado da solução;

m_2: Inserir vértice de maior economia: Utilizando-se o método *Add* Maior Economia, verifica para os vértices que ainda não estão na rota, o que possui a maior economia de inserção, e este é inserido na solução;

m_3: Trocar 2 vértices da solução: Escolhe-se 2 vértices que fazem parte da solução e estes são trocados de posição;

m_4: Retirar vértice: Escolhe-se aleatoriamente um vértice que faça parte da solução e este é removido dela;

m_5: Inserir vértice: Escolhe-se aleatoriamente um vértice que não faça parte da solução e este é inserido nela.

Através destes movimentos, várias estruturas de vizinhança foram criadas, conforme a seguir se relata.

3.3.4 Estruturas de Vizinhança

As estruturas de vizinhança são geradas a partir dos movimentos definidos anteriormente, resultando também em cinco estruturas.

A estrutura de vizinhança N^1 é gerada pelo movimento de se retirar o vértice de maior economia (m_1) e pode ser assim definida:

$$N^1(s) = \{s' : s + m_1\}$$

Um vizinho desta estrutura pode ser exemplificado conforme a figura 3.9.

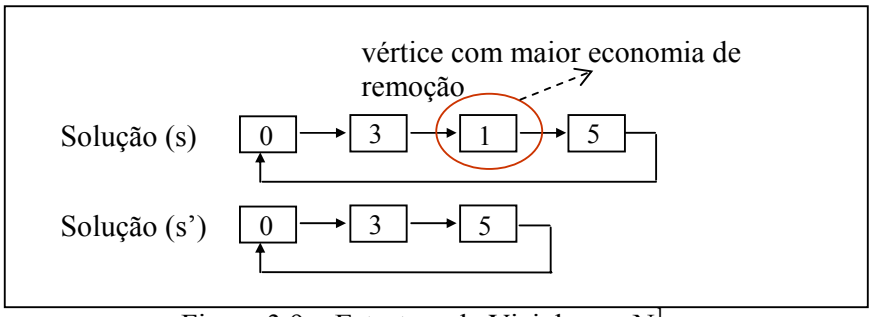

Figura 3.9 – Estrutura de Vizinhança N^1

A estrutura de vizinhança N^2 é gerada pelo movimento de inserir o vértice de maior economia (m_2) e pode ser assim definida:

$$N^2(s) = \{s' : s + m_2\}$$

Um vizinho desta estrutura pode ser exemplificado conforme a figura 3.10.

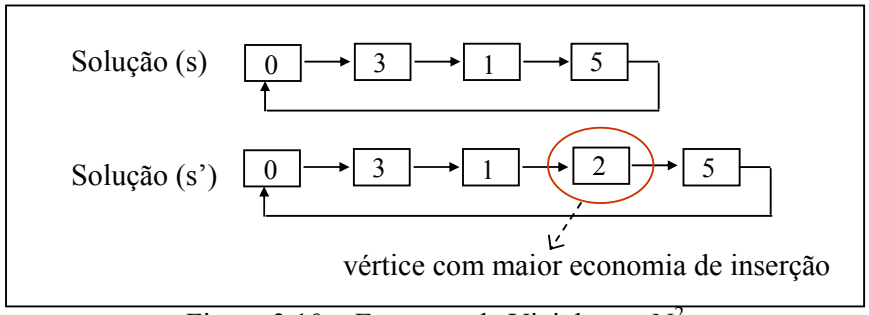

Figura 3.10 – Estrutura de Vizinhança N^2

A estrutura de vizinhança N^3 é gerada pelo movimento de trocar dois vértices da solução (m_3) e pode ser assim definida:

$$N^3(s) = \{s' : s + m_3\}$$

Um vizinho desta estrutura pode ser exemplificado conforme a figura 3.11.

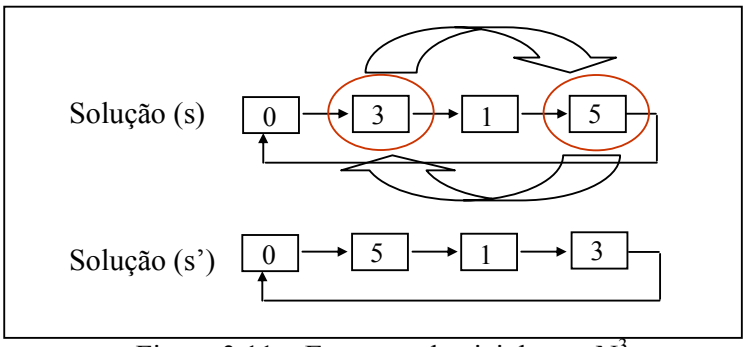

Figura 3.11 – Estrutura de vizinhança N^3

A estrutura de vizinhança N^4 é gerada pelo movimento de remover aleatoriamente um vértice da solução (m_4) e pode ser assim definida:

$$N^4(s) = \{s' : s + m_4\}$$

Um vizinho desta estrutura pode ser exemplificado conforme a figura 3.12.

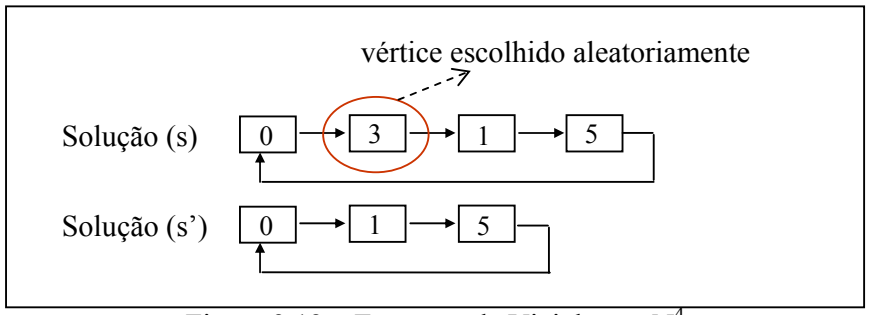

Figura 3.12 – Estrutura de Vizinhança N^4

A estrutura de vizinhança N^5 é gerada pelo movimento de inserir aleatoriamente um vértice na solução (m_5) e pode ser assim definida:

$N^5(s) = \{s' : s + m_5\}$

Um vizinho desta estrutura pode ser exemplificado conforme a figura 3.13.

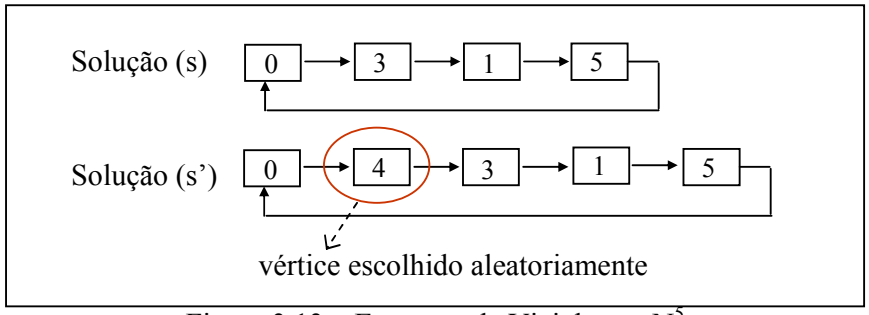

Figura 3.13 – Estrutura de Vizinhança N^5

3.3.5 Função de Avaliação

A função a ser minimizada, que avalia o problema do Caixeiro Viajante com Coleta de Prêmios, pode ser assim representada:

$$f(x) = \sum_{i \in N} \sum_{j \in N-(1)} c_{ij} x_{ij} + \sum_{i \in N} p_i (1 - y_i) + \alpha * \max \{0, - \sum_{i \in N} w_i * y_i + wo\}$$

Como se observa, esta função é composta pelo somatório dos custos de deslocamento na rota, pelo somatório das penalidades pagas nos vértices que não foram visitados, e ainda, por uma parcela que penaliza, através de um peso α, uma solução na qual o somatório dos prêmios coletados é menor que o prêmio mínimo pré-estabelecido.

As demais restrições foram contempladas através da representação adotada, ou seja, cada vértice só pode ser visitado no máximo uma vez, não sendo permitido a formação de ciclos na solução.

3.3.6 Algoritmo Proposto

Para resolução do problema do Caixeiro Viajante com Coleta de Prêmios utilizando uma modelagem heurística, foi proposto neste trabalho o algoritmo híbrido cujo pseudocódigo é descrito a seguir, o qual combina as metaheurísticas GRASP e VNS.

```
Procedimento Grasp+Vns
    s ← ∞ ;
    Enquanto número de execuções não satisfeito faça
        Construir a Solução Inicial s;
        Aplicar VNS/VND;
        Se ( f(s) < f(s*) ) então
            s* ← s
    Fim Enquanto
Fim Procedimento
```

Figura 3.14 – Algoritmo Proposto GRASP + VNS

Neste algoritmo o usuário define o número de execuções que ele deseja. A partir de então, executa-se a fase de construção, que procura construir uma solução viável e de qualidade. Logo em seguida é executada a fase de busca local, procurando refinar a solução inicial. Para isto, faz-se uso dos métodos de pesquisa em vizinhança variável VNS e VND. Após a fase de construção e refinamento da solução construída, atualiza-se a melhor solução encontrada sempre que esta for pior que a solução refinada.

A escolha do GRASP associado ao VNS deu-se porque ambos vêm obtendo bons resultados na resolução de diversos outros problemas combinatoriais.

3.3.7 Construção de uma Solução Inicial

O algoritmo da fase de construção foi baseado no algoritmo *Add* Maior Economia, diferenciando deste na escolha do elemento a ser inserido. Ao invés de inserir o elemento que oferecer a maior economia de inserção, uma lista de candidatos é criada a cada iteração, com os elementos que oferecerem as maiores economias de inserção, então um elemento desta lista é escolhido aleatoriamente e inserido na solução. Este algoritmo termina quando não existir mais economia positiva e o prêmio coletado seja maior que o prêmio mínimo pré-estabelecido.

Uma diferença em relação ao algoritmo da fase de contrução do GRASP básico, visto na figura 2.5, é que a cada geração de uma solução viável não é executado uma busca local para esta. Neste algoritmo proposto, executa-se a fase de construção *n* vezes e seleciona-se a melhor solução encontrada, aplicando-se nesta uma busca local. Isto permite ainda na fase de construção eliminar soluções iniciais de péssima qualidade.

Este algoritmo para construção de uma solução inicial viável pode ser descrito conforme a figura a seguir.

```
Procedimento ConstruirSoluçãoInicial
  s* ← ∞;
  Para j pertencente a V faça
    s ← Ø ;
    Inserir a origem em s
    Para todo k não pertencente a s faça
       Calcular a economia de inserção;
    Fim Para
    Enquanto não for atingido o prêmio mínimo ou
                  existir alguma economia positiva faça
       RLC = Lista dos elementos com maior economia;
       Selecione aleatoriamente, um elemento t ∈ RLC ;
       s ← s ∪{t};
       Atualizar Valores;
    Fim Enquanto
    Se f (s) < f (s*) então  s* ← s;
  Fim Para
  Retornar s;
Fim Procedimento
```

Figura 3.15 – Agoritmo para Construção de uma Solução Inicial

3.3.8 VNS Aplicado ao PCVCP

Os resultados obtidos através dos algoritmos de construção, não garantem necessariamente a obtenção de um ótimo local. Por isso, é sempre benéfica a fase de busca local, que por ser um procedimento de refinamento, procura uma melhoria na qualidade das soluções obtidas na fase de construção.

Neste trabalho implementou-se a metaheurística VNS, pois a qualidade das soluções obtidas nos procedimentos que utilizam apenas uma única estrutura de vizinhança pode ser comprometida se a estrutura escolhida não for adequada.

O algoritmo VNS aplicado ao problema pode ser descrito conforme a figura 3.16.

```
Procedimento VNS
    Seja s_0 a solução inicial gerada pela fase de construção GRASP;
    r = 7;
    s ← s_0 ;
    Enquanto tempo sem melhora < t faça
        k ← 1;
        Enquanto k ≤ r  faça
            Caso k:
                1: {s' : s + m_2}
                2: {s' : s + m_1}
                3: {s' : s + m_5}
                4: {s' : s + m_3 + m_3}
                5: {s' : s + m_4 + m_3 + m_4}
                6: {s' : s + m_5 + m_4}
                7: {s' : s + m_5 + m_5}
            Fim Caso
                s'' ← VND (s')
            Se f(s'') < f(s) então
                s ← s'';
                k ← 1;
            Senão
                k ← k + 1;
        Fim Enquanto
    Fim Enquanto
    Retorne s
Fim Procedimento;
```

Figura 3.16 – Algoritmo VNS aplicado ao PCVCP

Neste algoritmo parte-se de uma solução inicial gerada pelo algoritmo *ConstruirSoluçãoInicial*, apresentado na figura 3.15, e a cada iteração seleciona-se aleatoriamente um vizinho *s'* dentre as 7 estruturas de vizinhanças definidas. Esse vizinho é então submetido a um procedimento de busca local, no caso o VND. Se a solução ótima local, *s''*, for melhor que a solução *s* corrente, a busca continua de *s''* recomeçando da primeira estrutura de vizinhança. Caso contrário, continua-se a busca a partir da próxima estrutura de vizinhança. Este procedimento é encerrado quando o tempo de execução sem melhora ultrapassar *t* segundos.

3.3.9 VND Aplicado ao PCVCP

Após ter gerado um vizinho da solução corrente na heurística VNS, é importante realizar uma busca local encontrando o ótimo local para este vizinho. Novamente, escolheu-se uma heurística com várias estruturas de vizinhança visando escapar de ótimos locais ruins.

Para realizar a busca local faz-se uso de três heurísticas de refinamento, são elas: SeqDropSeqAdd, 2-*Optimal* e AddDrop. A cada iteração é escohida uma heurística e procura-se gerar um vizinho através de trocas na solução, aceitando-se somente soluções de melhora da solução corrente e retornando-se à primeira estrutura quando uma solução melhor é encontrada.

O pseudocódigo deste algoritmo é descrito a seguir.

```
Procedimento VND
    Seja s0 uma solução inicial;
    r = 3;
    s ← s0 ;
    k ← 1
    Enquanto k ≤ r  faça
        Caso k:
            1: SeqDropSeqAdd;
            2: 2-Optimal;
            3: AddDrop;
        Fim Caso
        Se f(s') < f(s) então
            s ← s' ;
            k ← 1;
        Senão
            k ← k + 1;
    Fim Enquanto
    Retorne s;
Fim Procedimento;
```

Figura 3.17 – Algoritmo VND Aplicado ao PCVCP

3.3.10 SeqDropSeqAdd aplicado ao PCVCP

Esta heurística consiste de uma seqüência de remoções, onde a cada iteração é retirado o vértice que fornecer a maior economia de remoção, seguidas de uma seqüência de inserções, onde a cada iteração é inserido o vértice que fornecer a maior economia de inserção. As remoções são baseadas no método *Drop* Maior Economia e as inserções no método *Add* Maior Economia. A figura a seguir descreve um algoritmo para esta heurística.

```
Procedimento SeqDropSeqAdd
    V = { s' ∈ N¹(s): f(s') < f(s) }
    Enquanto ( |V| > 0 ) faça
        Selecione o melhor s' ∈ V;
        s ← s' ;
        V = { s' ∈ N¹(s): f(s') < f(s) }
    Fim Enquanto
    V = { s' ∈ N²(s): f(s') < f(s) }
    Enquanto ( |V| > 0 ) faça
        Selecione o melhor s' ∈ V;
        s ← s' ;
        V = { s' ∈ N²(s): f(s') < f(s) }
    Fim Enquanto
    Retorne s;
Fim Procedimento;
```

Fugura 3.18 – Algoritmo SeqDropSeqAdd aplicado ao PCVCP

3.3.11 2-*Optimal* aplicado ao PCVCP

Esta heurística procura realizar possíveis trocas das arestas de uma rota, visando melhorar o valor da função objetivo através da mudança na seqüência em que os vértices são visitados. A figura a seguir ilustra um algoritmo para a heurística 2-*Optimal*.

```
Procedimento k-Optimal
   Seja s0 uma solução inicial;
   s ← s0;
   s* ← s;
   Repetir
      Q ← Lista com todos pares de arestas de R;
      Enquanto Q > Ø faça
         Z ← Lista com 2 arestas de Q;
         Remover as arestas Z de R gerando R';
         Construir todas possíveis rotas que contenham R';
         s* ← s que apresentar maior melhoria;
         Q ← Q - Z;
      Fim enquanto
      Se f(s*) < f(s) então
         s ← s*;
   Até f(s) ≥ f(s*)
   Retornar s;
Fim Procedimento
```

Figura 3.19 – Algoritmo 2-*Optimal* aplicado ao PCVCP

3.3.12 AddDrop aplicado ao PCVCP

Esta heurística consiste em inserir o vértice que fornecer a maior economia de inserção, baseado no método *Add* Maior Economia, mesmo que este piore a solução corrente, e remover o vértice que fornecer a maior economia de remoção, baseado no método *Drop* Maior Economia, também aceitando movimento de piora. A figura a seguir descreve um algoritmo para esta heurística.

```
Procedimento AddDrop
    V = { s' ∈ N¹(s) }
    Se ( |V| > 0 ) então
        Selecione o melhor s' ∈ V;
        s ← s';
    Fim Se
    V = { s' ∈ N²(s) }
    Se ( |V| > 0 ) então
        Selecione o melhor s' ∈ V;
        s ← s';
    Fim Se
    Retorne s;
Fim Procedimento;
```

Figura 3.20 – Algoritmo AddDrop aplicado ao PCVCP

4 Sistema Desenvolvido

4.1 Descrição do Sistema

O sistema desenvolvido, chamado "PCVCP Solver", tem o intuito de solucionar aproximadamente o Problema do Caixeiro Viajante com Coleta de Prêmios. Ele foi desenvolvido na linguagem C++ com a ferramenta C++ *Builder* 5.0, em virtude do poder e desempenho da linguagem e a disponibilidade de recursos da ferramenta.

O sistema procura oferecer uma certa flexibilidade ao usuário, permitindo a ele escolher o número de vértices que o problema terá, os arquivos com os dados que ele deseja executar e o número de execuções que ele deseja obter para o problema.

O funcionamento do sistema pode ser descrito em quatro etapas:

1. *Criar Estrutura de Dados*: o usuário define o número de vértices que o problema terá, e cria-se a estrutura de dados referente a estes vértices;
2. *Configurar a Estrutura de Dados*: o usuário escolhe quais serão os arquivos de dados utilizados na execução do problema, e a estrutura de dados é preenchida com os valores presentes nos arquivos;
3. *Configurar Número de Vezes de Execução*: o usuário define o número de vezes que ele deseja que seja executado o probema;
4. *Executar GRASP_VNS_VND*: o usuário executa o procedimento proposto neste trabalho GRASP_VNS_VND.

4.2 Telas do Sistema

1) Tela de criação da estrutura de dados

Nessa tela (representada pela Figura 4.1) o usuário seleciona o número de vértices que o problema terá, sendo criada a matriz de custos, os vetores de prêmios e penalidades, e a lista de Candidatos com os vértices do problema, aonde estes vão de 0 até n-1, sendo n o número de cidades passado pelo usuário.

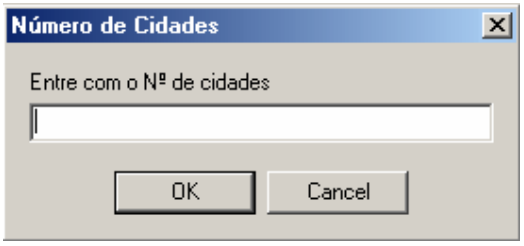

Figura 4.1 – Tela de Criação da Estrutura de Dados

2) Tela de entrada do arquivo que contém as distâncias entre os vértices

Nesta tela (representada pela Figura 4.2) o usuário seleciona o arquivo que contém as informações sobre os custos de deslocamento entre os vértices do problema. Estes dados são carregados para a matriz de custos.

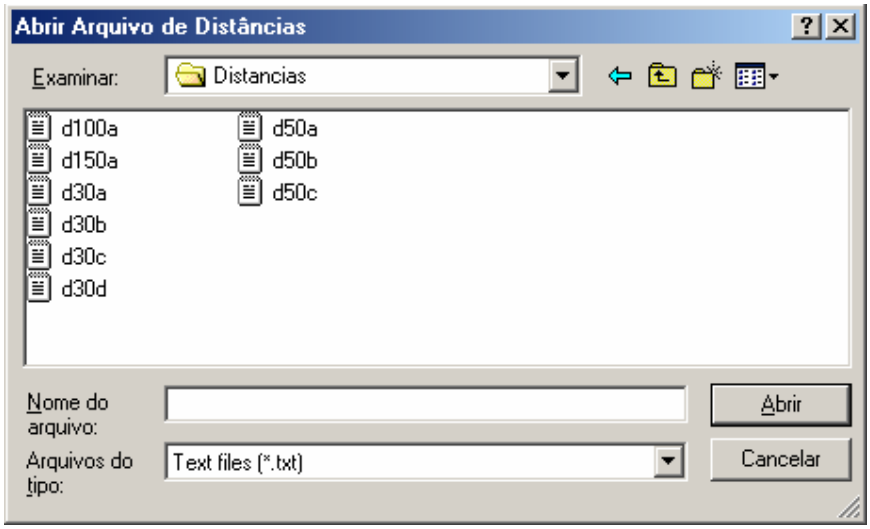

Figura 4.2 – Tela de entrada do arquivo de distâncias

3) Tela de entrada do arquivo que contém as penalidades de cada vértice

Nesta tela (representada pela Figura 4.3) o usuário seleciona o arquivo que contém as informações sobre as penalidades pagas caso o vértice não seja visitado. Estes dados são carregados para o vetor de penalidades.

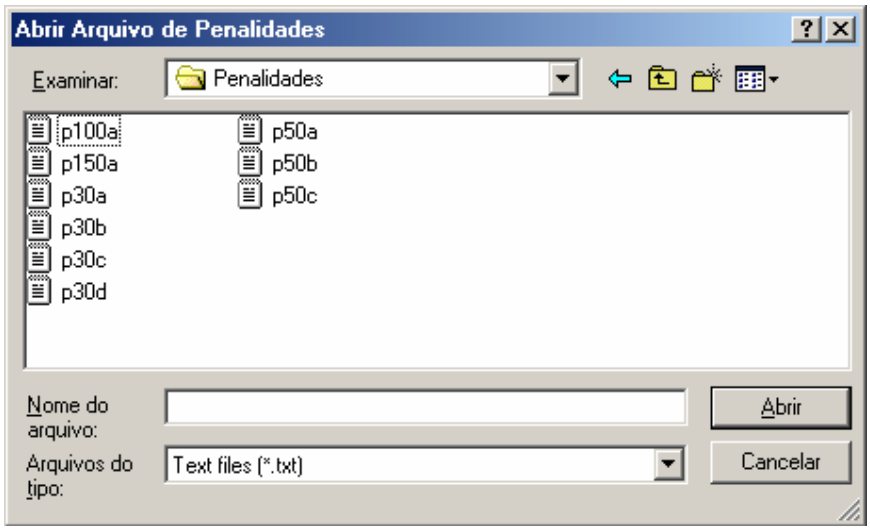

Figura 4.3 – Tela de entrada do arquivo penalidades

4) Tela de entrada do arquivo que contém os prêmios de cada vértice

Nesta tela (representada pela Figura 4.4) o usuário seleciona o arquivo que contém as informações sobre os prêmios coletados caso o vértice seja visitado. Estes dados são carregados para o vetor de prêmios.

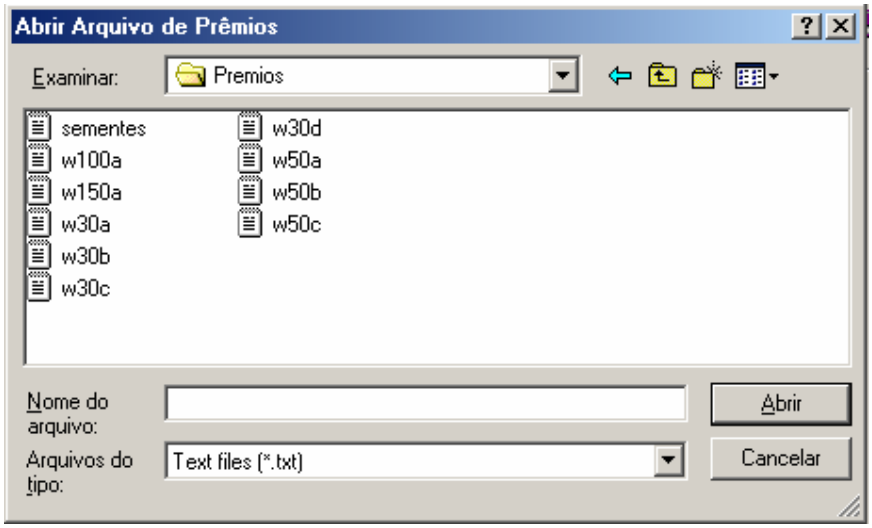

Figura 4.4 – Tela de entrada do arquivo prêmios

5) Tela de entrada do prêmio mínimo

Nesta tela (representada pela Figura 4.5) o usuário informa qual deve ser o prêmio mínimo a ser coletado para que uma solução não seja inviável.

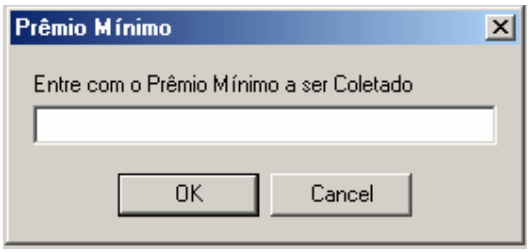

Figura 4.5 – Tela de entrada para o prêmio mínimo

6) Tela de entrada para o número de vezes de execução

Nesta tela (representada pela Figura 4.6) o usuário informa qual o número de vezes ele deseja que o sistema execute a instância do problema escolhida.

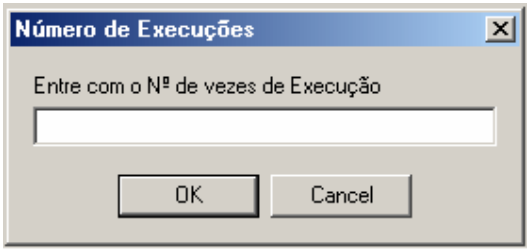

Figura 4.6 – Tela de entrada para o número de execuções

7) Tela de execução do programa

Nesta tela (representada pela Figura 4.7) o usuário visualiza os dados da execução, como o valor da função objetivo da solução gerada pela fase de construção, o valor da função

objetivo na fase de busca local, o tempo de execução, o tempo que faz que a solução não melhora, e o melhor valor da função objetivo encontrado.

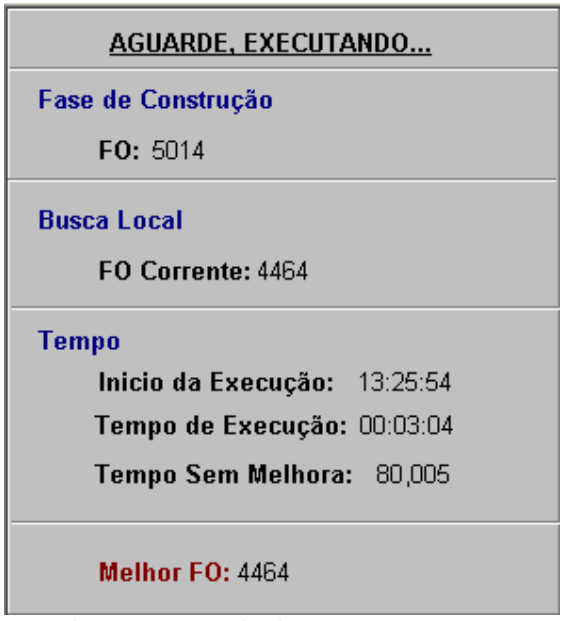

Figura 4.7 – Tela de Execução

8) Tela para salvar a solução encontrada

Nesta tela (representada pela Figura 4.8) o usuário salva a melhor solução encontrada em uma execução.

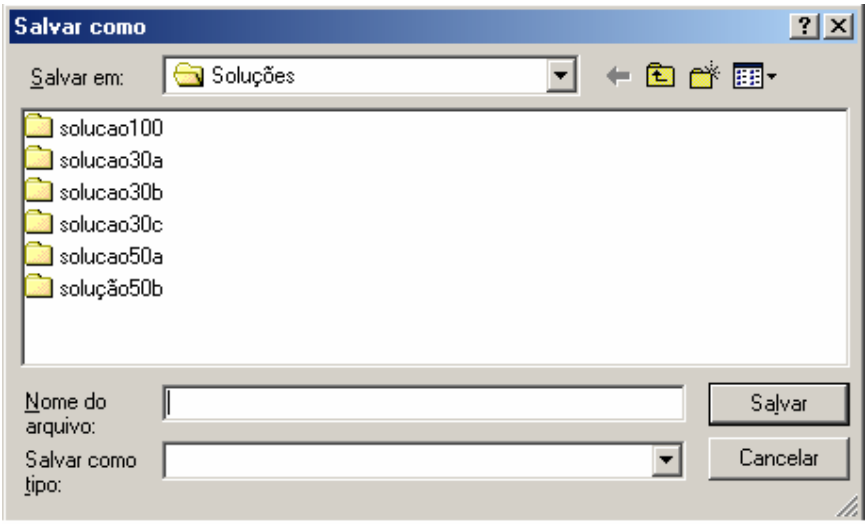

Figura 4.8 – Tela para Salvar a Solução Encontrada

9) Tela de visualização da solução encontrada

Nesta tela (representada pela figura 4.9) o usuário vizualiza a melhor solução encontrada na execução, observando os vértices que foram visitados, os que não foram, o valor da função objetivo, o prêmio coletado, a penalidade paga e a distância percorrida.

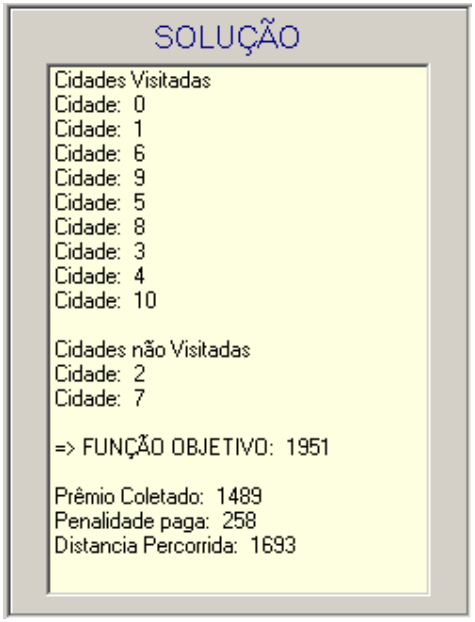

Figura 4.9 – Tela de Visualização da Solução

10) Tela principal do sistema

A partir dessa tela (representada pela Figura 4.10) o usuário seleciona as demais telas do sistema.

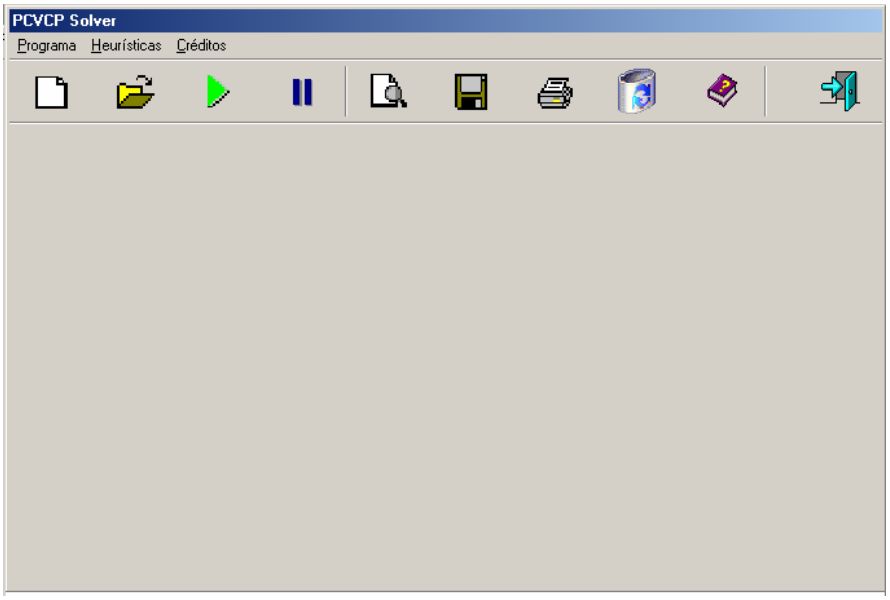

Figura 4.10 – Tela Principal do Sistema

5 Resultados Obtidos

O Problema do Caixeiro Viajante com Coleta de Prêmios, embora seja um problema com inúmeras aplicações, ainda é pouco explorado pela literatura afim. Ao que é de nosso conhecimento, dos poucos trabalhos associados ao PCVCP, não existe nenhuma biblioteca pública de problemas testes. Sendo assim, para avaliar a modelagem heurística proposta neste trabalho, instâncias foram obtidas da seguinte forma: as distâncias entre as cidades (vértices do grafo) foram geradas aleatoriamente no intervalo [50, 1000], o prêmio e a penalidade associados a cada uma dessas cidades também foram gerados aleatoriamente, considerando respectivamente os intervalos [1, 100] e [1, 750]. O prêmio mínimo a ser coletado representa 75% do somatório de todos os prêmios associados às cidades. A metodologia para a determinação dos prêmios e penalidades foi a utilizada no trabalho de Melo (2001). As instâncias utilizadas para teste neste trabalho encontram-se disponíveis em http://www.decom.ufop.br/prof/marcone/Orientacoes/OrientacoesConcluidas.htm.

Neste capítulo, apresentam-se os resultados computacionais obtidos com um significatico número de testes para instâncias geradas como mencionado anteriormente, e um comparativo entre os resultados encontrados nas modelagens exata e heurística.

Todos os algoritmos foram implementados na linguagem C++ e os testes computacionais foram realizados sob o Sistema Operacional Windows, e um microcomputador com processador Athlon XP de 1,53 Ghz e 256 MB de memória RAM.

5.1 Análise Comparativa dos Resultados Obtidos nas Modelagens Exata e Heurística

Esta seção apresenta os resultados obtidos nos testes para instâncias com até cinqüenta cidades. Assim como mencionado no Capítulo 3, os algoritmos propostos baseiam-se na metaheurística GRASP, consistindo de duas fases: construção da solução inicial e busca local na solução construída. A solução disponibilizada para a fase de busca local, na qual se aplica o algoritmo descrito pela Figura 3.15, é aquela que produz o menor valor da função de avaliação dentre um conjunto de soluções construídas pelo algoritmo da Figura 3.14. Esta metodologia de refinar a melhor solução construída é conhecida na literatura como GRASP com filtro.

Na Figura 5.1 tem-se um comparativo dos resultados obtidos nas modelagens exata e heurística. Na primeira coluna estão as instâncias do problema, na coluna 2, |V| representa a cardinalidade dos vértices que compõem a instância. A terceira e quarta coluna dizem respeito ao modelo exato, e representam respectivamente o tempo de execução e o valor do ótimo global. As colunas 5, 6 e 7 referem-se ao modelo heurístico, na quinta e sexta colunas encontram-se o tempo médio de execução e os melhores valores da função de avaliação (M-FO), conseguidos com o modelo heurístico, e na sétima coluna tem-se o desvio fornecido pelos vários testes realizados para as instâncias. O desvio pode ser descrito como a seguir:

$$\text{Desvio} = \left(\frac{FOMedia - FOMelhor}{FOMelhor} \right) \times 100$$

Comparação de Resultados entre o Modelo Exato e o Modelo Heurístico

		Modelo Exato		Modelo Heurístico		
Instância	**| V |**	**Tempo**	**Ótimo Global**	**Tempo**	**M-FO**	**Desvio**
v10	11	00:01	1765	00:01	1765	0 %
v20	21	01:05	2302	00:23	2302	0,36 %
v30a	31	01:26	3582	02:40	3582	0,79 %
v30b	31	01:40	2515	03:11	2515	1,05 %
v30c	31	29:46	3236	02:50	3236	0,65 %
v50a	51	180:00	Não encontrado	05:26	4378	1,89 %
v50b	51	180:00	Não encontrado	04:20	3945	1,75 %

Figura 5.1 – Comparativo entre os modelos Exato e Heurístico

Para cada instância foram realizados trinta testes, e apesar do modelo heurístico não garantir a otimalidade da solução final, este tem a capacidade de encontrar boas soluções a um custo computacional razoável, como pode ser observado na figura 5.1. Observa-se que para as intâncias até 30 vértices o ótimo global foi encontrado através do modelo exato, e o modelo heurístico também conseguiu encontrar o ótimo global, com um desvio pequeno em relação às soluções geradas. Este fato contribui para a validação do modelo heurístico proposto neste trabalho.

Observa-se também que para as instâncias v50a e v50b, as quais o modelo exato não conseguiu encontrar nenhuma solução viável em 3 horas de execução, o modelo heurístico encontra soluções viáveis em menos de 5 minutos. Apesar de não poder afirmar o quão perto estas soluções estão da solução ótima, o desempenho deste modelo para instâncias menores possibilita dizer que estas devem ser boas soluções.

5.2 Resultados acima de 50 Cidades

Esta seção apresenta os resultados obtidos nos testes para instâncias com mais de cinqüenta cidades. As instâncias verificadas foram: v100, v250, v500 e v1000, onde todas foram geradas aleatoriamente.

Na Figura 5.2 mostram-se os resultados encontrados para as instâncias com cardinalidade acima de cinqüenta. Na primeira coluna têm-se as instâncias para as quais se realizaram os testes. Na segunda coluna têm-se a cardinalidade (|V|) das instâncias. Na terceira coluna encontram-se os tempos médios (em minutos) gastos na execução do algoritmo proposto GRASP+VNS. A quarta coluna lista a melhor solução encontrada entre os testes realizados. A última coluna apresenta o desvio gerado nos testes, que é a difença entre a média das soluções geradas e a melhor solução encontrada, divido por esta.

Instância	| V |	Tempo	FO	Desvio
v100a	101	12:30	7645	1,54 %
v100b	101	11:23	7493	1,87 %
v250a	251	20:00	18036	1,18 %
v250b	251	14:00	17379	1,26 %
v500a	501	35:40	33000	0,67 %
v500b	501	40:10	33858	0,82 %
v1000	1001	145:00	63056	0,34 %

Figura 5.2 – Resultados obtidos para instâncias maiores que 50

A avaliação final para as instâncias entre 100 e 1000 cidades, é que as soluções foram geradas em um tempo relativamente pequeno, considerando a complexidade dos problemas. E o desvio também foi pequeno, mas não se sabe quão próximo estas soluções estão do ótimo global.

5.3 Fase de Construção x Fase de Busca Local

Esta seção apresenta uma análise comparativa entre a solução construída na fase de construção do GRASP e a solução refinada pelos métodos de pesquisa em vizinhança variável.

Fase de Construção x Fase de Refinamento

Instância	| V |	FC	FR	Melhora
v10	11	1765	1765	0 %
v20	21	2566	2302	10,3 %
v30a	31	4688	3582	23,6 %
v30b	31	3348	2515	24,9 %
v30c	31	4232	3236	23,6 %
v50a	51	5918	4378	26,1 %
v50b	51	5313	3945	25,8 %
v100a	101	9365	7645	18,4 %
v100b	101	9503	7493	21,2 %
v250a	251	20406	18036	11,7 %
v250b	251	19619	17379	11,5 %
v500a	501	35906	33000	8,1 %
v500b	501	37686	33858	10,2 %
v1000	1001	66410	63056	5,1 %

Figura 5.3 – Comparação dos resultados encontrados nas fases Construção e Refinamento

Na figura 5.3 tem-se uma análise comparativa entre os resultados encontrados na fase de construção e o encontrado no refinamento destas soluções. Na terceira coluna tem-se o valor da função de avaliação da solução encontrada na fase de construção GRASP. Na quarta

coluna tem-se o valor da função de avaliação da solução encontrada na fase de refinamento. E na última coluna, tem-se a percentagem de melhora da solução após ser refinada.

Observa-se que a fase de refinamento, utilizando as metodologias VNS e VND, obteve melhoras significativas nas soluções construídas inicialmente.

5.4 Comparação entre GRASP com filtro e sem filtro

Nesta seção procura-se mostrar a vantagem da utilização do filtro na fase de construção GRASP. Para isso, foram testadas as duas metodologias.

A figura 5.4 ilustra o resultado desses testes. A terceira e quarta colunas referem-se ao GRASP com filtro. Na terceira coluna tem-se o valor da função de avaliação da solução construída com filtro, enquanto na quarta coluna tem-se o valor da função de avaliação da solução refinada. A quinta e sexta colunas referem-se ao GRASP sem filtro. Na quinta coluna tem-se o valor da função de avaliação da solução construída sem filtro, enquanto na sexta coluna tem-se o valor da função de avaliação da solução refinada. Na última coluna tem-se o desvio em relação aos melhores resultados encontrados nas duas metodologias.

Influência de Filtro na fase de construção

		GRASP com filtro		GRASP sem filtro		
Instância	**| V |**	**FC-F**	**FR-F**	**FC-S**	**FR-S**	**Desvio**
v10	11	1765	1765	2916	1765	0 %
v20	21	2566	2302	3844	2302	0 %
v30a	31	4688	3582	5474	3582	0 %
v30b	31	3348	2515	4676	2515	0 %
v30c	31	4232	3236	5023	3290	1,66 %
v50a	51	5918	4378	7405	4628	5,71 %
v50b	51	5313	3945	6437	4182	6,01 %
v100a	101	9365	7645	10369	8183	7,03 %
v100b	101	9503	7493	11296	7809	4,21 %
v250a	251	20406	18036	21498	18958	5,11 %
v250b	251	19619	17379	19915	17885	2,91 %
v500a	501	35906	33000	36522	33416	1,26 %
v500b	501	37686	33858	38662	34527	1,97 %

Figura 5.4 – Comparação entre GRASP com e sem filtro

Observa-se que a metodologia GRASP com filtro sempre gera na fase de construção soluções melhores que GRASP sem filtro, o que acarreta em um menor tempo de execução, pois a fase de construção é mais rápida computacionalmente que a fase de refinamento. Observa-se também que a metodologia GRASP com filtro encontra soluções melhores, para instâncias elevadas do problema.

6 Conclusão

6.1 Análise dos resultados e conclusões

Apresenta-se neste trabalho algumas contribuições para a resolução do Problema do Caixeiro Viajante com Coleta de Prêmios.

Este trabalho faz uso de uma formulação matemática proposta por Egon Balas, para resolver o PCVCP de forma exata. E faz uso de metaheurísticas híbridas, combinando a metodologia GRASP (*Greedy Randomized Adaptive Search Procedures*) com as metodologias VNS (*Variable Neighborhood Search*) e VND (*Variable Neighborhood Descent*) para resolver o PCVCP de forma aproximada.

Na fase de construção foi notado que o uso da metodologia GRASP com filtro, isto é, escolher a solução que possui a menor função de avaliação dentre um subconjunto de soluções iniciais construídas, contribuiu significativamente na qualidade da busca local.

Na fase de busca local, pode-se justificar a escolha da estrutura de vizinhança: SeqDropSeqAdd (Seção 3.3.10) + 2-*Optimal* (Seção 3.3.11) + AddDrop (Seção 3.3.12), por esta gerar uma grande diversificação da solução encontrada na primeira vizinhança de busca (SeqDropSeqAdd). Na fase de construção, a solução inicial é determinada quando nenhuma melhora puder ser obtida com a inserção de uma nova cidade. Na maioria dos casos, isto acarreta um prêmio coletado relativamente superior ao prêmio mínimo pré-estabelecido, permitindo posteriormente que remoções sejam realizadas sem inviabilizar a nova solução. Logo, pode-se concluir que esta estrutura possui uma seqüência de vizinhanças que "favorece" uma maior redução na função de avaliação, o que permite em buscas heurísticas uma média de resultados de melhor qualidade.

Na comparação dos resultados encontrados nas modelagens exata e heurística, podemos verificar que os algoritmos heurísticos sempre atingiram o ótimo global para as instâncias onde este é conhecido.

Verifica-se também que a metaheurística híbrida proposta mostrou-se robusta, pois partindo de diferentes soluções iniciais chega-se a soluções finais que diferem da melhor solução encontrada em uma pequena percentagem.

Portanto, os resultados obtidos validam a utilização desta metaheurística híbrida para a resolução do Problema do Caixeiro Viajante com Coleta de Prêmios.

6.2 Sugestões para trabalhos futuros

Como trabalhos futuros podemos citar:

- Realizar o levantamento de dados reais, permitindo testar o sistema desenvolvido numa aplicação prática;
- Adaptar as contribuições aqui propostas para os demais problemas de otimização combinatória, onde a solução utiliza apenas um subconjunto de vértices do grafo;

- Implementar outras metodologias heurísticas, tais como Busca Tabu e Algoritmos Genéticos, para que se possa realizar uma comparação entre os resultados encontrados;
- Paralelizar os algoritmos aqui desenvolvidos;

7 Referências Bibliográficas

BALAS, Egon, The Prize Collecting Traveling Salesman Problem and Its Applications, *Management Science Research Report*, MSRR-664, (2001).

COSTA Jr., Aloísio Carlos T., *O Problema de Roteamento Periódico de Veículos: Uma abordagem via Metaheurística GRASP*, Dissertação de mestrado, Instituto de Computação, Universidade Federal Fluminense (UFF), Niterói, Rio de Janeiro (2003).

FEO, T.A. & RESENDE, M.G.C., Greedy randomized adaptive search procedures, *Journal of Global Optimization*, 6:109-133, (1995).

GLOVER, F., Future Paths for Integer Programming and links to Artificial Intelligence, *Computers and Operations Research*, 5:553-549, (1986).

GOLDBARG, M. C. & LUNA, H. P., *Otimização combinatória e programação linear: modelos e algoritmos*, 3ª Edição. Rio de Janeiro: Editora Campus, (2000).

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

KIRKPATRICK, S. et. al., Optimization by Simulated Annealing, *Science*, 220:671-680, (1983).

MARTINHON *et. al*., *An Hybrid GRASP+VNS Metaheuristic for the Prize-Collecting Traveling Salesman Problem*, Working Paper, Instituto de Computação, Universidade Federal Fluminense, Niterói (2000);

MCCULLOCH , W. S. & PITTS, W., A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5 (1943): 115-133;

MELO, Valdir A., *Metaheurísticas para o Problema do Caixeiro Viajante com Coleta de Prêmios*, Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro (2001);

MLADENOVIC, N. & HANSEN, P., Variable Neighborhood Search. *Computers and Operations Research*, 24:1097-1100, (1997).

ONG, H. L., Aproximate Algorithms for the Traveling Purchaser Problem, *Operations Research Letters* 1(5), 201-205 (1982).

PRADO, Darci, *Programação Linear* (Série Pesquisa Operacional, vol. 1). Belo Horizonte: Editora Desenvolvimento Gerencial, (1999).

RESENDE, Maurício G. C., A GRASP for Job Shop Scheduling, *AT&T Labs Research*, Florham Park, New Jersey, (1997).

SIMONETTI, Neil. Applications of a Dynamic Programming Approach to the Traveling Salesman Problem, (1998).

SOUZA, M. J. F. Inteligência Computacional para Otimização, Notas de aula, Departamento de Computação, Universidade Federal de Ouro Preto, disponível em http://www.decom.ufop.br/prof/marcone.

STÜTZLE, T. & DORIGO, M. I., ACO Algorithms for the Traveling Salesman Problem, Université Libre de Bruxelles, Belgium (1999).

TAILLARD, E. D., Ant Systems , Technical Report IDSIA-05-99, (1999).

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO – DECOM
CAMPUS UNIVERSITÁRIO – MORRO DO CRUZEIRO
CEP 35400-000 – OURO PRETO – MINAS GERAIS