Capítulo 1

Introdução

Este trabalho tem como objetivo propor, através de algoritmos aproximados, soluções para o Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP), referido na literatura como *Prize Collecting Traveling Salesman Problem* (PCTSP), (BALAS, 2001).

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 resolva 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.

Problemas desta natureza são comumente abordados através de heurísticas. Definimos 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 possuem 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.

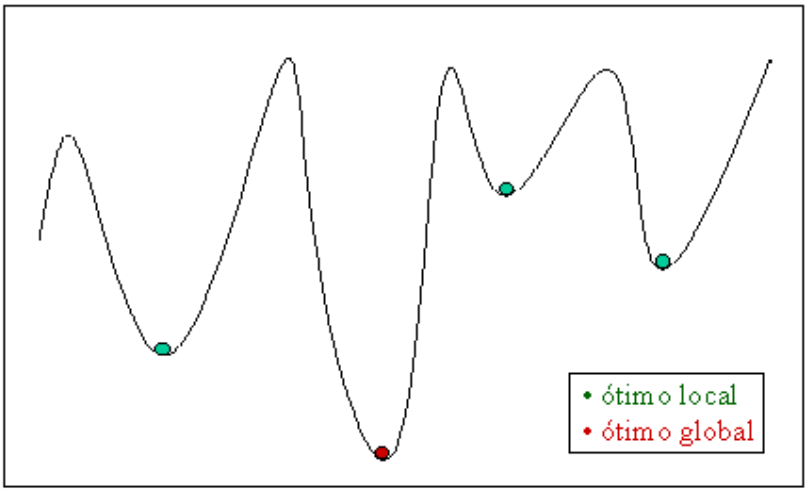

Figura 1 – Representação gráfica para um problema de minimização

Para a resolução do PCVCP, faz-se uso de conceitos de técnicas mais recentes, que tem 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.

No capítulo 2 descreve-se o problema do PCVCP, faz-se uma revisão de literatura sobre o tema e apresenta-se um modelo de Programação Linear Inteira, bem como sua implementação em um otimizador comercial.

No capítulo 3 apresenta-se uma descrição de heurística e de alguns métodos heurísticos utilizados na abordagem do problema.

No capítulo 4 apresenta-se uma descrição de metaheurística e de alguns métodos enquadrados como metaheurísticas a serem utilizados na abordagem do problema.

Capítulo 2

O Problema do Caixeiro Viajante com Coleta de Prêmios

2.1 Descrição do Problema

O PCVCP foi formulado inicialmente em 1985 por Ergon 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.

Uma solução para o PCVCP é demonstrada a seguir, considerando o grafo G com um custo de deslocamento entre seus vértices, um prêmio e uma penalidade para cada vértice (prêmio/penalidade). Constrói-se uma sub-rota a partir da origem (vértice 1) objetivando minimizar o custo e a penalidade, além de coletar no mínimo um prêmio w_0. Uma solução seria a sub-rota R_1 = {1,5,3,4,1}, isto é, o caixeiro sai da cidade 1 com um prêmio = 0,

constrói a sua rota passando pelas cidades 5, 3 e 4, coletando um prêmio = 28 (11 + 7 + 10), e retornando a origem, o caixeiro tem um custo de viajem = 58 (11 + 18 + 15 + 14), este também paga uma penalidade = 6, por não ter visitado a cidade 2.

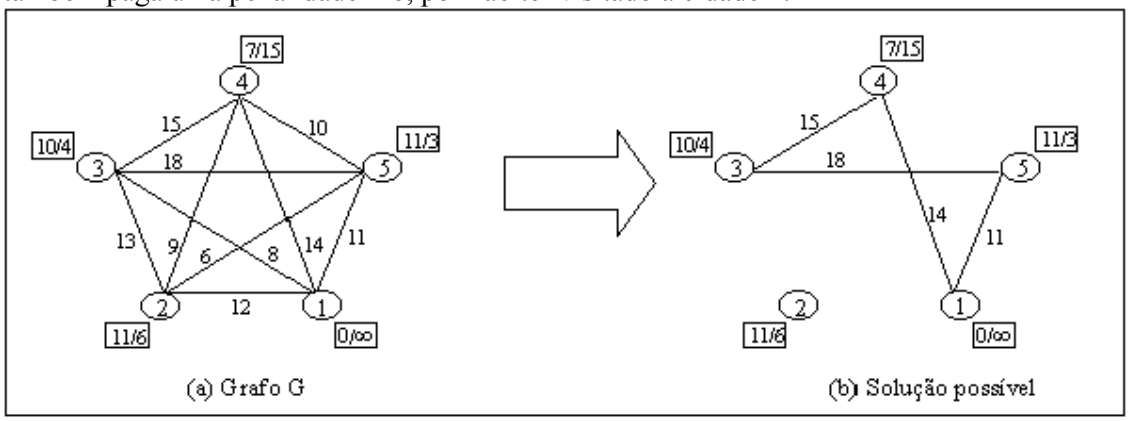

Figura 2 – Exemplo de uma solução para o PCVCP

2.2 Resenha na Literatura

As aplicações desenvolvidas para o PCVCP são poucas, apesar da grande aplicabilidade que este têm 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 distancia 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).

2.3 Modelo de Programação Linear Inteira

2.3.1 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 1 até $n = | N |$, sendo o vértice 1, sem perda de generalidade, assumido como depósito ou domicílio do caixeiro viajante. Para este vértice especial, será utilizado $w_1 = 0$ e $p_1 = \infty$. Segue uma formulação proposta em 1985 por Ergon Balas, Balas (2001), na qual as restrições 2.5 e 2.6 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:

(PCVCP) minimizar $\sum_{i \in N} \sum_{j \in N-\{1\}} c_{ij} x_{ij} + \sum_{i \in N} p_i (1 - y_i)$ (2.1)

sujeito à :

$$\sum_{j \in N-\{i\}} x_{ij} = y_i \qquad \forall\, i = 1,...,n \tag{2.2}$$

$$\sum_{i \in N-\{j\}} x_{ij} = y_j \qquad \forall\, j = 1,...,n \tag{2.3}$$

$$\sum_{i \in N} w_i * y_i \geq w_0 \tag{2.4}$$

$$\sum_{i \in N} \sum_{j \in N} f_{ij} - \sum_{i \in N} \sum_{j \in N} f_{ji} = y_i \quad \forall\, i = 1,...,n \;\; i \neq 1 \tag{2.5}$$

$$f_{ij} \leq (n-1) x_{ij} \qquad \forall\, (i,j) \in A \tag{2.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{2.7}$$

$$y_j \in \{0,1\} \qquad \forall\, i = 1,...,n \tag{2.8}$$

A restrição 2.2 diz que o somatório das arestas que sai do vértice *i* será igual a 1, se o vértice *i* estiver na rota e 0 caso contrário, a restrição 2.3 diz que o somatório das arestas que chegam ao vértice *j* será 1, se o vértice *j* estiver na rota e 0 caso contrário. A restrição 2.4 assegura que o prêmio coletado na rota será maior ou igual ao premio mínimo pré-estabelecido. As restrições 2.5 e 2.6 asseguram que *G'*(*y*, *x*) seja um ciclo. As restrições 2.7 e 2.8 asseguram a integralidade e não negatividade das variáveis *x* e *y*, respectivamente.

2.3.2 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 ou de primeiro grau e o resultado para as variáveis do modelo são valores reais ou contínuos. Este modelo também pode ser classificado como modelo de Programação Inteira, já que as variáveis somente admitem soluções inteiras (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 o PCVCP. Observa-se, entretanto, 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 demonstradas nas figuras a seguir.

Matriz de Custos				
Cidades	1	2	...	*n*
1	0	10	...	15
2	10	0	...	20
...	...	...	...	...
n	15	20	...	0

Cidades	1	2	...	*n*
Prêmio	0	5	...	30

Cidades	1	2	...	*n*
Penalidade	10000000	10	...	15

wo	120

Figura 3 - Entrada de dados para a resolução do PCVCP

Na figura 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.

Matriz Solução				
Cidades	1	2	...	*n*
1	0	1	...	0
2	0	0	...	1
3	...	...	...	...
n	1	0	...	0

Figura 4 - Saída de dados da resolução do PCVCP

Na figura 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 5, a seguir.

```
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;
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( matriz(i,j) : f(i,j) )
                   - @sum( matriz(i,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 5 – Implementação do modelo matemático

A validação do modelo se deu através de vários testes, onde se utilizaram 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 um modelo heurístico para a resolução do PCVCP, que como aplanado neste trabalho, busca encontrar boas soluções a um custo computacional razoável.

Execução da Modelagem Exata			
Nº Cidades	**Nº de interações**	**Tempo(min)**	**Ótimo Global**
6	155	00:01	94
10	7023	00:01	128
20	67500	00:12	206
30	235806	02:12	420
50	1015587	120:00:00	Não encotrado

Figura 6 - Resultados dos testes realizados

A figura 6 ilustra o número de cidades que o problema possui, o número de interações que o LINGO executou até encontrar o ótimo global, o tempo gasto para isto, e o valor do ótimo global para o PCVCP considerando os valores dos dados utilizados para testes, destacando que, para 50 cidades não só não se encontrou o ótimo global, como também não se encontrou nenhuma solução viável em 2 horas de execução.

Capítulo 3

Heurísticas

3.1 Descrição 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.

A seguir define-se alguns métodos de heurística que serão aplicadas neste trabalho. As duas primeiras (Add_Maior_Economia e Drop_Maior_Economia) são heurísticas construtivas e têm a finalidade de construir soluções buscando minimizar a função objetivo, e a terceira (*k-Optimal*) é um algoritmo de busca local utilizado para refinar uma solução, ou seja, tentar encontrar uma solução de melhor qualidade a partir de modificações efetuadas em outra solução.

3.2 Heurísticas Utilizadas

3.2.1 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:

$$\text{economia } (k) \;=\; g_k \;=\; \max\,(i,j)\;\{\, c_{ij} + p_k - c_{ik} - c_{kj} \,\},\;\; \forall\; k \notin R \qquad (4.1)$$

a qual é 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 ou a menor negativa (ou seja, $| g_k |$) 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 7, a seguir:

```
Procedimento Add_Maior_Economia
Inicialização
  R ← ∅;
  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
```

Figura 7 – Algoritmo de inserção Add_Maior_Economia

3.2.2 Drop_Maior_Economia

Este método também foi proposto por Melo (2001), onde partindo de uma solução que contenha todos os vértices, utiliza-se uma abordagem oposta a dos algoritmos de inserção, onde a cada iteração é retirado 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:

$$\text{economia}(k) = g_k = max\{ c_{ak,k} + c_{k,sk} - c_{ak,sk} - p_k \}, \forall k \in R \quad (4.2)$$

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 alguma economia positiva e o somatório de todos os prêmios dos vértices pertencentes à rota for igual ou maior que o prêmio mínimo pré-estabelecido.

O método Drop_Maior_Economia pode ser descrito conforme a figura 8, 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 ≥ w_0) então
        Remover o Vértice k de R;
        Atualizar valores;
    Senão
        Continua ← falso;
    Fim Se
  Fim Enquanto
```

Figura 8 – 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.

3.2.3 k-*Optimal* ou k-Substituições

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 objetiva, 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 a 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 9, a seguir.

```
Procedimento k-Optimal
    Seja s0 uma solução inicial;
    s ← s0;
    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 ;
```

Figura 9 – Algoritmo *k-Optimal*

Capítulo 4

Metaheurísticas

4.1 Descrição de Metaheurística

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 míopes, 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, 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.

A seguir descrevem-se algumas destas metaheurísticas que serão utilizadas neste trabalho.

4.2 Metaheurísticas Utilizadas

4.2.1 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 a seguir.

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

Figura 10 – 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. A componente probabilística 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. Uma algoritmo da fase de construção do GRASP pode ser descrito conforme a figura a seguir.

Procedimento *Construção GRASP*
$x \leftarrow \emptyset$;
Inicializar o conjunto de candidatos C;
Enquanto $C \neq \emptyset$ **faça**
$t_i = min\ \{g(t) \mid t \in C\}$;
$t_s = max\ \{g(t) \mid t \in C\}$;
RLC={ $t \in C \mid g(t) \leq t_i + \alpha\ (t_s - t_i)$ };
Selecione aleatoriamente, um elemento $t \in$ RLC ;
$x \leftarrow x \cup \{s\}$;
Atualizar o conjunto de candidatos C;
Fim Enquanto
Retornar x ;

Figura 11 – 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. Neste trabalho serão utilizadas as metaheurísticas VNS e VND para realizar a fase de busca local, sendo estas explicadas mais adiante.

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 em 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.

4.2.2 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 s0 uma solução inicial;
    Seja r o número de estruturas diferentes de vizinhaça;
    s ← s0;
    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 Se
        Fim Enquanto
    Fim Enquanto
    Retorne s ;
```

Figura 12 – 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.

4.2.3 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 vizinhaça;
$s \leftarrow s_0$;
$k \leftarrow 1$
Enquanto $k \leq r$ **faça**
Encontre o melhor vizinho $s' \in N^{(k)}(s)$;
Se $f(s') < f(s)$ **então**
$s \leftarrow s'$;
k $\leftarrow 1$;
Senão
$k \leftarrow k + 1$;
Fim Se
Fim Enquanto
Retorne s ;

Figura 13 – Algoritmo VND

Conclusão

Este trabalho propõe a resolução do problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP). Como uma primeira etapa fez-se a implementação do modelo exato do problema, verificando que tal problema só consegue resolver problemas de pequenas dimensões. Para uma segunda etapa, a ser implementada em Projeto Orientado II, propõe-se a utilização de uma metodologia heurística híbrida para a resolver o problema de forma mais expressiva. Com soluções iniciais sendo geradas através da metaheurística GRASP, sendo as melhores soluções construídas refinadas por um procedimento de busca local, no caso o VNS/VND.

Bibliografia

BALAS, Ergon, 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).

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).

Índice