MÁRIO TVRZSKÝ DE GOUVÊA

Engenheiro Naval, Escola Politécnica da USP, 1988

ALGORITMOS PARA A RESOLUÇÃO DE UM PROBLEMA GERAL DE ROTEAMENTO DE MÚLTIPLOS VEÍCULOS

Dissertação apresentada à Escola Politécnica da USP para a obtenção do Título de Mestre em Engenharia.

São Paulo, 1992

Orientador: **Prof. Dr. Marco Antonio Brinati**
Professor Titular do Departamento de Engenharia Naval da EPUSP

A Meus Pais

RESUMO

O presente trabalho tem por objetivo propor algoritmos, facilmente implementáveis em microcomputadores, capazes de dimensionar uma frota de veículos para atender um conjunto de pontos de demanda, ou capazes de alocar uma frota existente a esse conjunto, de forma que nenhum ponto de demanda deixe de ser atendido em um período de tempo especificado.

Os algoritmos propostos são capazes de trabalhar com tipos de veículos de capacidades e velocidades diferentes, considerando os custos fixo e variável de cada tipo. A seleção e alocação dos veículos a viagens é feita visando-se a minimização do custo da frota, podendo os veículos realizar mais de uma viagem no período de tempo máximo especificado. Os algoritmos permitem que mais de uma viagem seja realizada para pontos cuja demanda total não possa ser atendida por nenhum tipo de veículo disponível em uma única viagem.

Os algoritmos desenvolvidos apresentaram, em problemas clássicos e menos abrangentes de minimização de distâncias ou custos variáveis, como o problema básico de roteamento de veículos, resultados piores que os de outros algoritmos especificamente desenvolvidos para tais casos, mas podem ser utilizados, sem alterações estruturais, em problemas reais associados ao cenário bem mais abrangente do problema geral de roteamento de múltiplos veículos adotado no presente trabalho, para minimização do custo total da frota.

RESUMO

O presente trabalho tem por objetivo propor algoritmos, fácilmente implementáveis em microcomputadores, capazes de dimensionar uma frota de veículos para atender um conjunto de pontos de demanda, ou capazes de alocar uma frota existente a esse conjunto, de forma que nenhum ponto de demanda deixe de ser atendido em um período de tempo especificado.

Os algoritmos propostos são capazes de trabalhar com tipos de veículos de capacidades e velocidades diferentes, considerando os custos fixo e variável de cada tipo. A seleção e alocação dos veículos a viagens é feita objetivando-se a minimização do custo da frota, podendo os veículos realizar mais de uma viagem no período de tempo máximo especificado. Por último, os algoritmos permitem que mais de uma viagem seja realizada para pontos cuja demanda total não possa ser atendida por nenhum tipo de veículo disponível em uma única viagem.

Os algoritmos desenvolvidos, apesar de apresentarem, em problemas clássicos de minimização de distâncias ou custos variáveis pouco restritos, como o problema básico de roteamento de múltiplos veículos, resultados inferiores a outros algoritmos, especificamente desenvolvidos para os mesmos, podem ser utilizados, sem aletrações estruturais, em problemas reais baseados no cenário muito mais restrito do problema geral de roteamento de múltiplos veículos, que objetiva a minimização do custo da frota, adotado no presente trabalho.

ABSTRACT

The present study proposes algorithms, of easy microcomputer implementation, which can obtain a vehicle fleet, or place an existing one, to serve a set of demand centers, leaving no demand center without serving, in a specified period of time.

The proposed algorithms can work with different vehicle types, varying on velocity and capacity, considering fixed and variable costs of each type. The selection and placing of vehicles to voyages objectifies minimization of fleet cost, being possible for vehicles making more than one voyage on the specified period of time. The proposed algorithms also allow more than one voyage per demand center, if the demand is bigger than the capacity of any type of vehicle available.

The proposed algorithms have a worse performance on solving low-restricted distance or variable cost minimization problems, like the basic multiple routing problem, than other specificly developed algorithms, but can be used to solve, without structural changes, real problems based on more restricted, fleet cost minimization objectived, general multiple vehicle routing scenario, proposed on following study.

ÍNDICE

1- Apresentação da Dissertação 1

1.1- Introdução 1

1.2- Motivação da Dissertação 2

1.3- Objetivo da Dissertação 4

1.4- Estrutura da Dissertação 6

1.5- Nomenclatura Adotada na Dissertação 7

2- Revisão Bibliográfica 10

2.1- Introdução 10

2.2- Modelos de Programação Linear para o Problema do Caixeiro Viajante 10

2.3- Métodos Heurísticos para o Problema do Caixeiro Viajante 14

2.3.1- Heurísticas para a Construção de Roteiros 14

2.3.2- Heurísticas para a Melhoria de Roteiros 16

2.3.3- Heurísticas Compostas 17

2.4- Modelos de Programação Linear para o Roteamento de Múltiplos Veículos 17

2.5- Métodos Heurísticos para Roteamento de Múltiplos Veículos 21

2.5.1- Algoritmo de Varredura 23

2.5.2- Algoritmo de Clarke & Wright 25

2.5.3- Algoritmo de Partição do Roteiro Gigante 27

2.5.4- Algoritmo de Fisher & Jaikumar 31

2.6- Métodos Heurísticos para o Problema "Dial-a-Ride" 35

2.6.1- Algoritmo ADARTW 36

2.7- Critérios para a Comparação da Eficiência de Heurísticas 37

3- Descrição do Cenário Padrão Adotado 40

3.1- Introdução 40

3.2- Obtenção do Cenário Padrão 40

3.3- O Cenário Padrão do Problema Geral de Roteamento de Múltiplos Veículos 43

3.4- A Frota de Veículos 45

3.5- Os Problemas Gerais de Dimensionamento e de Alocação de Frota 46
3.6- Considerações Finais 47

4- Algoritmo para o Problema de Dimensionamento de Frota 48
4.1- Introdução 48
4.2- Algoritmo para o Problema de Dimensionamento de Frota 50
4.3- Detalhamento dos Principais Passos do Algoritmo 52
- Decomposição de Pontos com Demanda Superior à Capacidade do Maior Modelo de Veículo Disponível 52
- Criação do Roteiro Gigante 54
- Determinação do Número de Iterações a Serem Realizadas 55
- Criação de Rede Orientada com Fator Redutor de Custo Fixo 56
- Partição do Roteiro Gigante 59
- Combinação de Viagens 59
- Pós-Processamento da Solução (Partição de Novo Roteiro Gigante 62
4.4- Comentários Finais 63

5- Algoritmo para o Problema de Alocação de Frota 64
5.1- Finalidade do Algoritmo 64
5.2- Formulação do Algoritmo 64
5.3- Algoritmo para o Problema de Alocação de Frota 66
5.4- Desenvolvimento do Algoritmo 69
- Critério de Ordenação de Pontos de Demanda 70
- Critério de Ordenação dos Veículos da Frota 70
- Critério de Alocação de Pontos de Demanda aos Veículos da Frota 74
- Procedimento de Melhoria de Alocações 75
5.5- Comentários Finais 76

6- Algoritmo de Melhoria de Soluções 77
6.1- Introdução 77

6.2- Formulação do Algoritmo de Melhoria de Soluções 78
6.3- Algoritmo de Melhoria de Soluções 79
6.4- Comentários Finais 80

7- Testes Computacionais dos Algoritmos 82
7.1- Introdução 82
7.2- Primeira Série de Testes: Cenário Pouco Restrito 83
7.3- Segunda Série de Testes: Cenário Restrito 90
7.3.1- Introdução 90
7.3.2- Variação do Número de Pontos de Demanda 93
7.3.3- Variação da Demanda Média 95
7.3.4- Variação do Desvio Padrão da Demanda 97
7.3.5- Variação da Posição Relativa da Base ao Centro Geométrico dos Pontos de Demanda 99
7.3.6- Variação da Duração Máxima de Operação dos Veículos 101
7.3.7- Variação da Distância Média dos Pontos de Demanda ao seu Centro Geométrico 103
7.3.8- Melhoria da Solução Obtida 105
7.4- Considerações Finais 109

8- Conclusão 113
8.1- Introdução 113
8.2- Conclusões 114
8.3- Recomendações para Novos Trabalhos 115

Bibliografia 118

Apêndice 122
I- Programa de Dimensionamento de Frota 122
II- Programa de Alocação de Frota 151
III- Programa de Melhoria de Soluções 177
IV- Gerador de Problemas Aleatórios 185

1- APRESENTAÇÃO DA DISSERTAÇÃO

1.1- INTRODUÇÃO

"Transportar consiste em movimentar algo de um lugar para outro".

O transporte está intimamente relacionado com a sociedade humana há muito tempo. As grandes construções, o comércio, a manufatura, tudo envolve transporte em maior ou menor grau, sendo o custo de transporte uma parcela importante no custo de qualquer produto, direta ou indiretamente.

Uma das maneiras mais efetivas de reduzir-se os custos de transporte é através de um dimensionamento e alocação eficientes de uma frota de veículos.

Em problemas de transporte de grande porte, envolvendo o atendimento de dezenas, ou até centenas, de pontos por uma frota de veículos, são necessários algoritmos para o dimensionamento e alocação de uma frota, sendo muito difícil conseguir isso de maneira eficiente usando apenas a experiência, lápis e papel.

Desde a década de 60, problemas reais de dimensionamento e alocação de frota são objeto de pesquisa, visando a obtenção de algoritmos.

A adaptação desses algoritmos a problemas diversos dos originais, entretanto, é, muitas vezes, difícil, ou mesmo impossível, devido à necessidade de incorporar-se certas características que não eram relevantes no problema que gerou esses procedimentos.

1.2- MOTIVAÇÃO DA DISSERTAÇÃO

A idéia de realizar-se a presente dissertação de mestrado surgiu durante realização de projeto no Centro de Estudos e Pesquisas do Departamento de Engenharia Naval da EPUSP (CEPEN-USP), intitulado "Sistema de Transporte de Pessoal para as Plataformas da Bacia de Campos" [3].

O objetivo do projeto era de dimensionar um sistema aquaviário para transporte de passageiros para as plataformas de petróleo operando na bacia de Campos, estado do Rio de Janeiro, em substituição do sistema em utilização na época, majoritáriamente realizado por helicópteros. O projeto envolvia a escolha do tipo de embarcações e o dimensionamento da frota a ser utilizada para atender a demanda de transporte de passageiros, incluindo a definição do roteamento das embarcações, a um custo mínimo.

As características do problema, com plataformas bastante distanciadas da costa, e com transbordo demorado, associadas ao tipo de embarcações consideradas, com grande capacidade de passageiros, poderiam levar a viagens de duração muito grande, esbarrando em um dos requisitos do projeto, que especificava que as viagens não excedessem uma duração máxima, visando o conforto dos passageiros.

Com isso, além do número de passageiros em uma embarcação não poder ser superior à sua capacidade de transporte, por motivos de segurança e conforto, as viagens realizadas por cada veículo deveriam ter a sua duração limitada, considerando-se o tempo gasto com o veículo em movimento, navegando entre a base e as plataformas atendidas, e com o veículo parado ao largo das plataformas, embarcando e desembarcando passageiros nas mesmas.

Com o desenvolvimento do trabalho constatou-se que nenhum dos algoritmos disponíveis na literatura podia ser utilizado diretamente para a resolução do problema, devido principalmente à não consideração de uma restrição de duração máxima de viagens, que no caso era fundamental. Com isso, foi necessário adaptar-se diversos algoritmos, incluindo-se uma restrição de duração máxima para as viagens, para permitir a definição dos parâmetros do veículo a ser utilizado (velocidade e capacidade) e a obtenção dos roteiros dos veículos da frota.

Durante o desenvolvimento desse trabalho, percebeu-se que o problema tinha ainda muito campo para pesquisa, podendo ser objeto de uma dissertação de mestrado, pois o problema de transporte de passageiros para as plataformas de petróleo da bacia de Campos possuía implícitas simplificações que permitiram a fácil adaptação de algoritmos comuns da literatura, como:

- A demanda de transporte de passageiros de qualquer uma das plataformas de petróleo era muito menor que a capacidade de qualquer um dos tipos de veículos considerados para realizarem as viagens;
- Cada plataforma de petróleo era atendida uma única vez por semana, o que permitiu que se definisse uma janela de uma semana para o cenário, na qual cada plataforma era atendida uma única vez por um único veículo;
- A frota deveria ser uniforme, o que simplificou em muito o processo de obtenção da frota mais econômica e de alocação dessa frota em viagens diárias;
- A inexistência de uma frota anteriormente em operação que pudesse ser utilizada para o problema permitiu partir-se "do zero", sem que o processo de escolha e alocação da frota tivesse que considerar veículos já existentes;

as quais, se não existissem, impediriam que os algoritmos pesquisados pudessem ser utilizados no trabalho.

Pensando-se no fato de que em problemas reais nem sempre existiriam fatores, como ocorreu no problema de transporte de passageiros para as plataformas de petróleo da bacia de Campos, que reduziriam os mesmos a um caso clássico da literatura, permitindo a aplicação quase direta de variada gama de algoritmos existentes, definiu-se como objeto de estudo da presente dissertação a formulação de algoritmos que pudessem resolver problemas mais abrangentes, de forma que os algoritmos pudessem resolver variada gama de problemas reais sem serem necessárias alterações estruturais nos mesmos.

1.3- OBJETIVO DA DISSERTAÇÃO

O objetivo da presente dissertação é a proposição de procedimentos que permitam a resolução de problemas reais, baseados no problema geral de roteamento de múltiplos veículos, definido na presente dissertação, sem serem necessárias alterações estruturais nos mesmos.

Para tanto, partindo-se da análise de algoritmos existentes, desenvolveram-se heurísticas que permitem o dimensionamento de uma frota (ou alocação de uma frota existente) composta por veículos de velocidades e capacidades diferentes, com base nos custos fixo e variável de cada modelo disponível, com a inclusão de uma duração máxima para as viagens, e com a possibilidade de existirem pontos com demanda de transporte tal que sejam necessários mais de um veículo para atendê-la. Também foi incluída a possibilidade de um veículo realizar mais de uma viagem, mantendo-se no limite da duração máxima permitida.

Inicialmente considerou-se apenas o dimensionamento, ou definição, de uma frota a partir de uma variada gama de tipos diferentes de veículos, disponíveis em número ilimitado. Entretanto, com o avanço da dissertação, decidiu-se considerar também o caso de já existir uma frota de veículos de tipos diferentes, com número conhecido de veículos de cada tipo, para atender os pontos de demanda do problema, passando esse problema, distinto do problema de dimensionamento de frota, a ser chamado de problema de alocação de frota.

Por último, desenvolveu-se um algoritmo de melhoria de soluções, que tem por objetivo procurar reduzir o custo das soluções obtidas do problema geral de roteamento de múltiplos veículos.

Foram desenvolvidos os algoritmos doravante chamados:

Algoritmo de dimensionamento de frota: dimensiona e aloca uma frota de veículos, escolhidos dentre uma série de modelos diferentes fornecidos.

Algoritmo de alocação de frota: aloca uma frota fornecida de veículos, sem alterar a sua composição.

Algoritmo de melhoria de soluções: procura melhorar soluções obtidas pelos algoritmos de dimensionamento e de alocação de frota fornecidas, reduzindo o seu custo.

Foram desenvolvidas na dissertação implementações computacionais para os três algoritmos, em linguagem Turbo-Pascal. A opção pelo microcomputador foi possível graças ao baixo tempo de processamento dos algoritmos desenvolvidos, conforme objetivos iniciais da dissertação.

Evitou-se, por isso, o uso de métodos exatos de programação matemática, que necessitariam de um processamento em separado de partes do algoritmo, utilizando-se programas comerciais. Essa opção, se por um lado pode diminuir a qualidade da solução obtida, torna o uso dos algoritmos facilitado a quaisquer usuários em potencial, a um baixo custo.

1.4- ESTRUTURA DA DISSERTAÇÃO

A presente dissertação é apresentada em oito capítulos e um apêndice.

- O capítulo primeiro é uma introdução à dissertação.

- O capítulo segundo é o resultado da revisão bibliográfica realizada, no qual são apresentados e discutidos conceitos e metodologias, utilizados nos capítulos subseqüentes.

- O capítulo terceiro define o cenário dos algoritmos desenvolvidos, ou seja, as características básicas que os problemas reais devem ter para poderem ser resolvidos pelos mesmos.

- O capítulo quarto detalha o algoritmo desenvolvido para o problema de dimensionamento de frota.

- O capítulo quinto detalha o algoritmo desenvolvido para o problema de alocação de frota.

- O capítulo sexto detalha o algoritmo desenvolvido para melhoria de soluções.

- O capítulo sétimo apresenta os resultados dos testes computacionais aos quais foram submetidas as implementações computacionais dos algoritmos apresentados e desenvolvidos ao longo da dissertação.

- O capítulo oitavo é uma síntese dos principais pontos do trabalho, sendo apresentadas suas contribuições, limitações, comentários sobre sua utilização prática e sugestões para sua continuidade em novos trabalhos.

- O apêndice é composto por quatro partes, cada qual detalha e apresenta a implementação computacional dos algoritmos de dimensionamento e alocação de frota, de melhoria de soluções e do programa de geração de cenários.

1.5- NOMENCLATURA ADOTADA NA DISSERTAÇÃO

Os símbolos a seguir são utilizados com o mesmo significado em toda a dissertação. Quando é usada a palavra nó, entende-se um nó qualquer do problema (inclusive a base de operações da frota), diferentemente da expressão ponto de demanda, que não engloba a base.

N:	Número de pontos de demanda;
NV:	Número de veículos disponíveis para atendimento de pontos de demanda;
NTV:	Número de modelos diferentes de veículos disponíveis para serem utilizados para o atendimento de pontos de demanda;
d_{ij}:	Distância entre os nós número i e número j;

Vs_K:	Velocidade de serviço do veículo k (ou do veículo de modelo k);
C_K:	Capacidade do veículo k (ou do veículo de modelo k);
Cf_K:	Custo fixo do veículo k (ou do veículo de modelo k) por período do tempo equivalente à duração máxima de jornada;
Co_K:	Custo variável do veículo k (ou do veículo de modelo k) por unidade de distância em movimento (custo variável);
t_{IJK}:	Tempo que o veículo k (ou um veículo de modelo k) leva para dirigir-se do nó i ao nó j;
t_i:	Duração do abastecimento, na base, da carga destinada ao ponto de demanda i, somada à duração da descarga da mesma no referido ponto;
D_i:	Demanda do ponto de demanda i;
TC_I:	Taxa de carregamento ou descarga no nó i ;
TV:	Período de tempo no qual todos os pontos de demanda deverão ser atendidos por pelo menos um veículo, doravante chamado de duração de jornada;
R_i:	Número do nó que ocupa a i-ésima posição em um roteiro gigante;
F:	Fator redutor de custo fixo;
min:	Menor dentre diversos números;
max:	Maior dentre diversos números;
resto:	Resto de uma divisão;
trunc:	Maior inteiro menor ou igual a um valor;

Para o cenário-padrão, apresentado no capítulo 3, adotou-se a seguinte nomenclatura:

μp_i:	Demanda média dos pontos de demanda;
σp_i:	Desvio-padrão da demanda dos pontos de demanda;
μd_0:	Distância da base ao centro geométrico do conjunto dos pontos de demanda;
μd_i:	Distância média dos pontos de demanda ao centro geométrico de seu conjunto;

Para os testes computacionais adotou-se:

NVG:	Número de viagens;
CT:	Custo total da solução obtida;
TP:	Tempo de processamento;
μC:	Capacidade média da frota;
μV:	Velocidade média da frota;

2- REVISÃO BIBLIOGRÁFICA

2.1- INTRODUÇÃO

Na revisão bibliográfica são abordados seis assuntos:

a) Modelos de programação linear para o problema do caixeiro viajante;

b) Métodos heurísticos para o problema do caixeiro viajante;

c) Modelos de programação linear para roteamento de múltiplos veículos;

d) Métodos heurísticos para roteamento de múltiplos veículos;

e) Métodos heurísticos para o problema "dial-a-ride";

f) Critérios para a comparação da eficiência de heurísticas;

Nesse capítulo será apresentada uma descrição detalhada dos principais algoritmos utilizados durante a elaboração da dissertação, e que sejam relevantes à mesma, sendo um dos objetivos do mesmo servir de guia de referência, fornecendo ao leitor não apenas um resumo das características de cada metodologia apresentada, mas também uma descrição do seu funcionamento.

2.2- MODELOS DE PROGRAMAÇÃO LINEAR PARA O PROBLEMA DO CAIXEIRO VIAJANTE

Caixeiros viajantes são indivíduos que tem por missão colocar os produtos de um estabelecimento comercial em locais fora de sua praça. Para tanto percorrem diversas cidades, saindo da cidade base (onde se

encontra sediado o estabelecimento comercial), procurando vender, em cada uma, os seus produtos, retornando, por fim, à primeira cidade.

O problema do caixeiro viajante deriva do cotidiano dos mesmos, e pode ser apresentado da seguinte forma:

"Qual a melhor seqüência em que diversas cidades devem ser percorridas para que, partindo-se de uma cidade, percorra-se todas as cidades uma única vez, de forma que o tempo (ou distância ou custo) gasto na locomoção seja mínimo?"

A modelagem apresentada a seguir foi extraída da coletânea de Bodin et alii [2]. d_{IJ} é a distância entre os nós número i e número j, x_{IJ} é uma variável binária (1 se o caixeiro, partindo do nó i, dirije-se a seguir ao nó j, 0 em caso contrário) e S é um conjunto de restrições, dentre algum dos apresentados a seguir, para evitar o aparecimento de sub-ciclos na solução, isto é, seqüências fechadas de nós que não incluam a base.

$$\text{minimizar} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} \cdot x_{ij} \qquad [2.1]$$

sujeito a

$$\sum_{i=1}^{N} x_{ij} = 1 \qquad j=1,..,N \qquad [2.2]$$

$$\sum_{j=1}^{N} x_{ij} = 1 \qquad i=1,..,N \qquad [2.3]$$

$$X = \{ x_{ij} \} \in S \qquad [2.4]$$

$$x_{ij} = \begin{cases} 0 \\ 1 \end{cases} \qquad i,j=1,..,N \qquad [2.5]$$

Deve-se garantir que, de qualquer nó i, saia apenas um único arco, que tenha como destino um nó j, e que a cada nó i, chegue um único arco, que tenha partido de um nó k, sendo isso expresso nas equações [2.2] e [2.3]. Dessa forma, entretanto, não se evita a formaçao de sub-ciclos, sendo, por isso, necessária a equação [2.4], onde S pode ser expresso nas formas alternativas [2.6], [2.7] ou [2.8].

$$S = \{ x_{ij} \} : \sum_{i \in Q} \sum_{j \notin Q} x_{ij} \geq 1 \qquad [2.6]$$

, para qualquer subconjunto Q de N não vazio

$$S = \{ x_{ij} \} : \sum_{i \in R} \sum_{j \in R} x_{ij} \leq | R | - 1 \qquad [2.7]$$

, para qualquer subconjunto não vazio R de {2,3,..,N}

$$S = \{ x_{ij} \} : y_i - y_j + (N.x_{ij}) \leq (N - 1) \qquad [2.8]$$

, para $2 \leq (i \neq j) \leq N$, para números reais y_i.

Tanto a equação [2.6] quanto a [2.7] geram aproximadamente 2^N restrições. O número de restrições geradas pela equação [2.8] é, entretanto, de apenas $N^2 - 3N + 2$. A figura 2.1, extraída de Bodin et alii [2], que representa uma solução com dois sub-ciclos, será utilizada para ilustrar o funcionamento das três formulações.

FIGURA 2.1 - DOIS SUB-ROTEIROS

A equação [2.6] garante que qualquer subconjunto próprio, não vazio, Q de nós deve estar conectado aos outros nós da rede na solução X. Na figura 2.1, a solução apresentada vai de encontro à equação [2.6] quando Q contiver os nós {1,2,3}.

A formulação [2.7], que funciona de maneira diversa, resulta na não existência de sub-ciclos em X, já que um ciclo, no subconjunto de nós R, deve necessáriamente possuir um número de arcos igual ao número de nós. Na figura 2.1, a solução apresentada vai de encontro à formulação [2.7] quando R={4,5,6}.

Já a formulação [2.8] gera uma contradição sempre que a matriz X contenha um sub-ciclo. Aplicando-se essa formulação aos arcos 4-5, 5-6 e 6-4 na figura 2.1, e somando-se as restrições, obtém-se $3n \leq 3(n-1)$, o que é uma contradição.

Outras modelagem podem ser formuladas para o problema do caixeiro viajante, como a proposta por Gavish & Graves, apresentada no trabalho de Bodin et alii [2]. O elevado número de restrições e de variáveis exigido pelas modelagens existentes de programação linear, bem como o elevado tempo requerido pelos pacotes computacionais existentes, tornam demorada a solução exata do problema do caixeiro viajante. Bodin et alii [2] apresentam uma metodologia exata para o problema do caixeiro viajante, proposta inicialmente por Held & Karp, que se mostrou satisfatória para a resolução de problemas com mais de 300 nós.

2.3- MÉTODOS HEURÍSTICOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE

As heurísticas para o problema do caixeiro viajante são agrupadas em três classes (Bodin et alii [2]):

- Heurísticas para a construção de roteiros;
- Heurísticas para a melhoria de roteiros;
- Heurísticas compostas;

2.3.1- HEURÍSTICAS PARA A CONSTRUÇÃO DE ROTEIROS

As heurísticas para a construção de roteiros criam o mesmo a partir de um conjunto de n nós, isto é, determinam um caminho para, partindo da base, percorrer-se os n nós, não necessitando, para tanto, de nenhuma solução inicial.

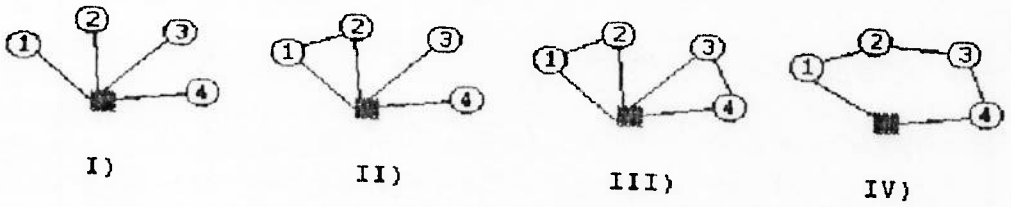
Serão apresentadas três das heurísticas para construção de roteiros mais utilizadas. Será chamado de base o nó imposto para ser o início do roteiro.

a) Vizinho mais próximo: Escolher o nó inicial mais próximo da base, e inserir o mesmo na primeira posição do roteiro, conectando-se o mesmo à base. Escolher sucessivamente os nós mais próximos ao último nó conectado, e colocar os mesmos na última posição do roteiro, até que todos os nós tenham sido incluídos.

b) Método de Clarke & Wright: Calcular a economia s_{IJ} (= $d_{I0} - d_{IJ} + d_{J0}$) para todos os pares de nós i-j. Ordenar essas economias decrescentemente, e, seguindo essa ordem, incorporar no roteiro as

viagens i-j, até que todos os nós do problema façam parte do mesmo. O método de Clarke & Wright é representado na figura 2.2. Supondo que as economias sejam, em ordem decrescente, s_{12} ($=s_{21}$), s_{34} ($=s_{43}$), s_{23} ($=s_{32}$), seguidas pelas seis economias s_{IJ} restantes, conectamos os nós 1 e 2 (obtendo três viagens), depois os nós 3 e 4 (obtendo duas viagens), e, finalmente, os nós 2 e 3, unindo as duas viagens em uma única.

FIGURA 2.2 – FUNCIONAMENTO DA HEURÍSTICA DE CLARKE & WRIGHT



Obs: supor que S12 > S34 > S23

c) Inserção mais próxima: Achar o nó i mais próximo à base e conectar o mesmo à base, formando o roteiro (base - nó i - base). Encontrar o nó k, não pertencente à sub-rota, mais próximo a qualquer dos nós da sub-rota. Achar o arco (i - j) da sub-rota que minimize a expressão ($d_{IK} + d_{KJ} - d_{IJ}$), e inserir o nó k entre os nós i e j. Repetir o procedimento até a inclusão de todos os nós.

Diversos outros métodos são apresentados por Bodin et alii [2], como o de inserção de menor custo e o método da envoltória convexa.

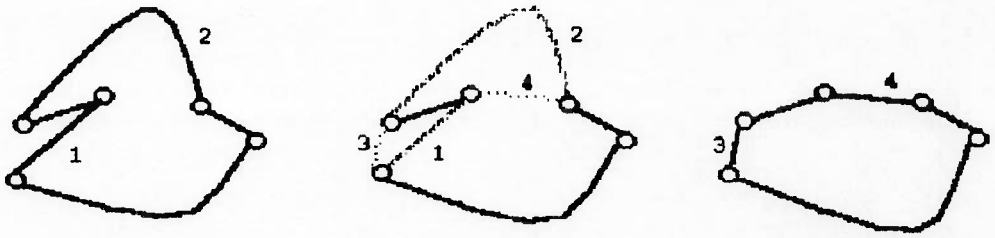
2.3.2 HEURÍSTICAS PARA A MELHORIA DE ROTEIROS

As heurísticas para a melhoria de roteiros necessitam de um roteiro inicial qualquer, para, através de procedimentos combinatórios, melhorarem o mesmo.

Essas heurísticas geralmente se baseiam em procedimentos combinatórios, sendo a melhoria no roteiro feita através da troca simultânea de k arcos da rota original. Isso é feito eliminando-se k arcos do roteiro e conectando-se os nós "soltos" a outros nós através de outros k arcos. Se com uma troca ocorrer uma melhoria do roteiro, a mesma será efetivada. Esse novo roteiro será, a seguir, submetido ao mesmo processo de melhoria, até que o roteiro fornecido não possa mais ser melhorado com a troca simultânea de k arcos. A otimalidade da solução só é garantida por esse procedimento se for imposto k=N, sendo N o número de nós do problema, pois todos os possíveis roteiros serão analisados. Esse procedimento combinatório ótimo, chamado N-ótimo, é recomendado apenas para roteiros que contenham poucos nós, sendo muito demorado.

Usualmente faz-se k=2 (sendo a heurística chamada procedimento 2-ótimo) ou k=3 (sendo a heurística chamada procedimento 3-ótimo). O conceito 2-ótimo é extensivamente utilizado na presente dissertação, não apenas para a resolver-se o problema do caixeiro viajante, mas também nos algoritmos desenvolvidos para dimensionamento, para alocação de uma frota fornecida, e para melhoria de soluções. A figura 2.3 ilustra a troca simultânea de dois arcos.

FIGURA 2.3 – PROCEDIMENTO DE MELHORIA PELA TROCA SIMULTÂNEA DE DOIS ARCOS



Roteiro original

Substituição dos arcos 1 e 2 pelos arcos 3 e 4

Roteiro melhorado

2.3.3 HEURÍSTICAS COMPOSTAS

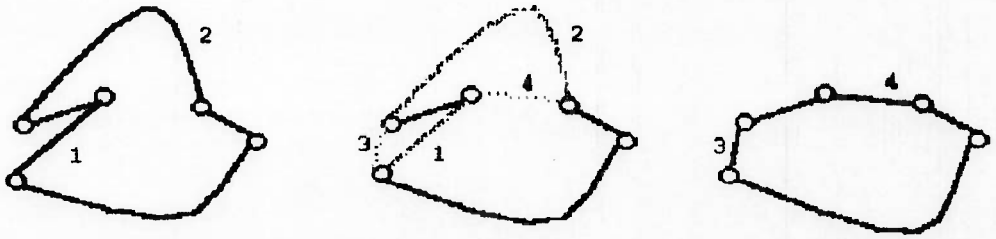
As heurísticas compostas são um conjunto integrado por uma heurística para a construção de roteiros e por uma (ou mais) heurísticas para a melhoria de roteiros.

Quaisquer heurísticas existentes podem ser utilizadas. Uma combinação recomendada, que alia boas soluções finais a um baixo tempo de processamento, é a utilização da heurística do vizinho mais próximo para criar um roteiro inicial, e a heurística 2-ótimo para melhorar essa solução. Pode-se, também, melhorar a solução obtida pela heurística 2-ótimo utilizando-se a heurística 3-ótimo.

2.4- MODELOS DE PROGRAMAÇÃO LINEAR PARA O ROTEAMENTO DE MÚLTIPLOS VEÍCULOS

O problema de roteamento de múltiplos veículos deriva do problema do caixeiro viajante.

FIGURA 2.3 – PROCEDIMENTO DE MELHORIA PELA TROCA SIMULTÂNEA DE DOIS ARCOS



Roteiro original

Substituição dos arcos 1 e 2 pelos arcos 3 e 4

Roteiro melhorado

2.3.3 HEURÍSTICAS COMPOSTAS

As heurísticas compostas são um conjunto integrado por uma heurística para a construção de roteiros e por uma (ou mais) heurísticas para a melhoria de roteiros.

Quaisquer heurísticas existentes podem ser utilizadas. Uma combinação recomendada, que alia boas soluções finais a um baixo tempo de processamento, é a utilização da heurística do vizinho mais próximo para criar um roteiro inicial, e a heurística 2-ótimo para melhorar essa solução. Pode-se, também, melhorar a solução obtida pela heurística 2-ótimo utilizando-se a heurística 3-ótimo.

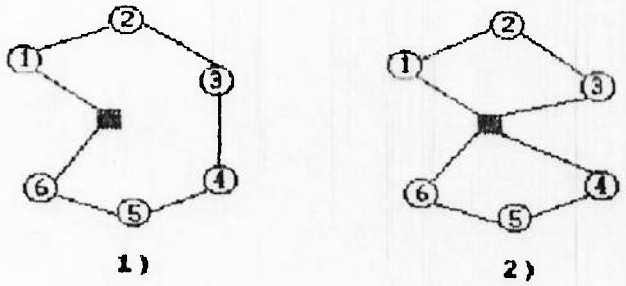
2.4- MODELOS DE PROGRAMAÇÃO LINEAR PARA O ROTEAMENTO DE MÚLTIPLOS VEÍCULOS

O problema de roteamento de múltiplos veículos deriva do problema do caixeiro viajante.

Define-se viagem como sendo uma seqüência fechada de nós, dos quais um é a base. No problema do caixeiro viajante, obtém-se sempre uma única viagem, pois o caixeiro viajante parte de uma cidade, percorre todas as cidades e retorna à cidade inicial.

No problema de roteamento de múltiplos veículos, dispõe-se de NV veículos, sendo que cada qual deve realizar uma única viagem, e todos os nós, exceto a base, devem estar incluídos em uma única viagem. A distância total percorrida deve ser mínima. A figura 2.4 mostra uma solução de um problema com um e dois veículos.

FIGURA 2.4 - SOLUÇÃO COM UM E DOIS VEÍCULOS



O problema básico de roteamento de múltiplos veículos considera, também, que em cada nó seja feita a coleta (ou entrega) de uma quantidade conhecida de um único tipo de produto, cada veículo possuindo um limite superior para a quantidade máxima transportada na viagem (restrição de capacidade dos veículos). Pode-se incluir uma restrição para que a duração máxima das viagens, incluíndo-se o tempo gasto com a coleta dos produtos, não exceda uma duração predeterminada (restrição de duração das viagens), originando o problema básico de roteamento de múltiplos veículos com restrição de duração.

A modelagem para problemas básicos de roteamento de múltiplos veículos com restrição de duração apresentada a seguir foi adaptada do

trabalho de Bodin et alii [2], sendo que $x_{i,jv}$ é uma variável binária que assume o valor 1 se o veículo v viaja do nó i ao nó j, e 0 em caso contrário.

$$\text{minimizar} \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{v=1}^{NV} c_{ij} \cdot x_{ijv} \qquad [2.9]$$

sujeito a

$$\sum_{i=0}^{N} \sum_{v=1}^{NV} x_{ijv} = 1 \qquad (j=1,..,N) \qquad [2.10]$$

$$\sum_{j=0}^{N} \sum_{v=1}^{NV} x_{ijv} = 1 \qquad (i=1,..,N) \qquad [2.11]$$

$$\sum_{i=0}^{N} x_{ikv} - \sum_{j=0}^{N} x_{kjv} = 0 \qquad \begin{pmatrix} v=1..NV \\ k=0..N \end{pmatrix} \qquad [2.12]$$

$$\sum_{j=1}^{N} x_{0jv} \leq 1 \qquad (v=1,..,NV) \qquad [2.13]$$

$$\sum_{i=1}^{N} x_{i0v} \leq 1 \qquad (v=1,..,NV) \qquad [2.14]$$

$$\sum_{i=1}^{N} d_i \left(\sum_{j=0}^{N} x_{ijv} \right) \leq C_v \qquad (v=1,..,NV) \qquad [2.15]$$

$$\sum_{i=1}^{N} t_i \left(\sum_{j=0}^{N} x_{ijv} \right) + \sum_{i=0}^{N} \sum_{j=0}^{N} t_{ijv} \cdot x_{ijv} \leq TV \qquad (v=1,..,NV) \qquad [2.16]$$

$$x_{ijv} = \begin{cases} 0 \\ 1 \end{cases} \qquad (\forall\ i,j,v) \qquad [2.17]$$

$$X = \sum_{v=1}^{NV} x_{ijv} \in S = \bigcup_{v=1}^{NV} S_v \qquad [2.18]$$

$$S_v = \left\{ x_{ijv} : \sum_{i \in Q} \sum_{j \in Q} x_{ijv} \leq | Q | - 1 \right\} \qquad [2.19]$$

para qualquer subconjunto não vazio Q de {1,2,..,N}

, onde c_{IJ} é o custo da viagem do nó i ao nó j (no trabalho de Bodin et alii [2] o mesmo independe do tipo de veículo que a realiza) NV é o número de veículos, C_v a capacidade do veículo v, TV é a duração máxima de jornada dos veículos, d_I é a demanda no nó i (d_0=0), t_{IJV} tempo que o veículo v leva para saindo do nó i atingir o nó j (t_{ii}=∞) e t_I a duração do abastecimento, na base, da carga destinada ao nó i, somada à duração da descarga da mesma no referido ponto.

A função objetivo da equação 2.9 visa minimizar o custo total de viagem.

As equações 2.10 e 2.11 garantem, respectivamente, que exatamente um veículo chega e sai de cada nó j (j=1,...,n). A equação 2.12 assegura que o veículo que chega e sai do nó j é o mesmo. Desde que as equações 2.10 e 2.12 implicam a validade da equação 2.11, esta é redundante e pode ser eliminada.

As equações 2.13 e 2.14 asseguram que a disponibilidade de veículos não é excedida; também para o nó 0, as equações 2.12 e 2.13 implicam a validade da equação 2.14, que, sendo redundante, pode ser suprimida.

A equação 2.15 representa a restrição de capacidade do veículo (admite-se que a quantidade de produto a ser embarcada em qualquer nó j possa ser atendida por um único veículo).

A equação 2.16 assegura que a duração total de qualquer viagem não ultrapasse a duração máxima permitida para os veículos.

O conjunto S da equação 2.18, que tem por objetivo evitar o aparecimento de sub-roteiros para cada veículo, pode ser, por exemplo, constituído por restrições impostas para cada veículo v, isto é, S pode ser expresso como a união de conjuntos Sv definidos pela equação 2.19 para todo subconjunto não vazio Q (1,2,...,N), onde |Q| representa o número de nós do conjunto Q.

Devido ao grande número de restrições geradas pelas formulações apresentadas, a solução exata é inviável para a maioria dos problemas reais, o que gerou a necessidade de outras metodologias, com solução não necessáriamente ótima, para a sua resolução.

2.5- MÉTODOS HEURISTICOS PARA ROTEAMENTO DE MÚLTIPLOS VEÍCULOS

As heurísticas para o problema de roteamento de múltiplos veículos podem ser divididas em seis grupos (Bodin et alii [2]):

- Heurísticas de agrupamento e roteamento;
- Heurísticas de roteamento e partição;
- Heurísticas de economia e inserção;
- Heurísticas de melhoria;

- Métodos baseados em programação matemática;
- Métodos de otimização interativa.

As heurísticas do primeiro grupo realizam um agrupamento dos nós em viagens, e, posteriormente, roteirizam os nós de cada viagem, utilizando procedimentos de resolução do problema do caixeiro viajante.

As heurísticas do segundo grupo funcionam de maneira oposta. Criam um "roteiro gigante" unindo todos os nós, utilizando procedimentos de resolução do problema do caixeiro viajante, e, a seguir, particionam esse roteiro em viagens, de forma a atender a todas as restrições.

As heurísticas do terceiro grupo consideram inicialmente uma viagem exclusiva a cada nó, e procuram agrupar os nós de duas viagens em uma única viagem, funcionando de maneira similar ao procedimento de Clarke & Wright para construção de roteiros, detalhado anteriormente.

As heurísticas do quarto grupo são baseadas nas heurísticas para melhoria de roteiros do problema do caixeiro viajante, utilizando técnicas combinatórias para melhorar uma dada solução para o problema do roteamento de multiplos veiculos.

As heurísticas do quinto grupo utilizam métodos exatos para resolver subproblemas contidos na heurística, como, por exemplo, a alocação de nós a viagens.

As heurísticas do sexto grupo exigem um interação constante com um operador experimentado, capaz de analisar uma solução parcial e de tomar decisões, influindo no andamento da heurística. A solução, nessas

heurísticas, será tanto melhor quanto maior for a experiência do operador.

Serão apresentadas, a seguir, algumas das heurísticas mais conhecidas para a resolução de problemas de roteamento de múltiplos veículos.

2.5.1- ALGORITMO DE VARREDURA

Esse algoritmo foi desenvolvido por Gillett & Miller [13] na década de 70. É uma heurística de agrupamento e roteamento, que realiza inicialmente uma partição dos nós em viagens e, posteriormente, realiza um roteamento dos nós em cada viagem. O algoritmo de varredura foi proposto para a resolução de problemas de grande porte (com mais de 100 nós) e foi desenvolvido para problemas básicos de roteamento de múltiplos veículos de um mesmo tipo (capacidades iguais).

O funcionamento do algoritmo de varredura é o seguinte:

1) Fixar um sistema de coordenadas polares com origem na base e calcular as coordenadas de cada nó;
2) Criar uma lista de nós, na qual os mesmos estão arranjados em ordem crescente de ângulo;
3) Fixar como nó inicial o primeiro nó da lista;
4) Partindo de um nó inicial, agrupar em uma viagem nós consecutivos, não alocados a nenhuma viagem, até que a capacidade do veículo seja excedida (exclue-se da viagem esse último nó que viola a restrição);
5) Realizar o roteamento dos nós contidos nessa viagem, através de uma heurística para o problema do caixeiro viajante;

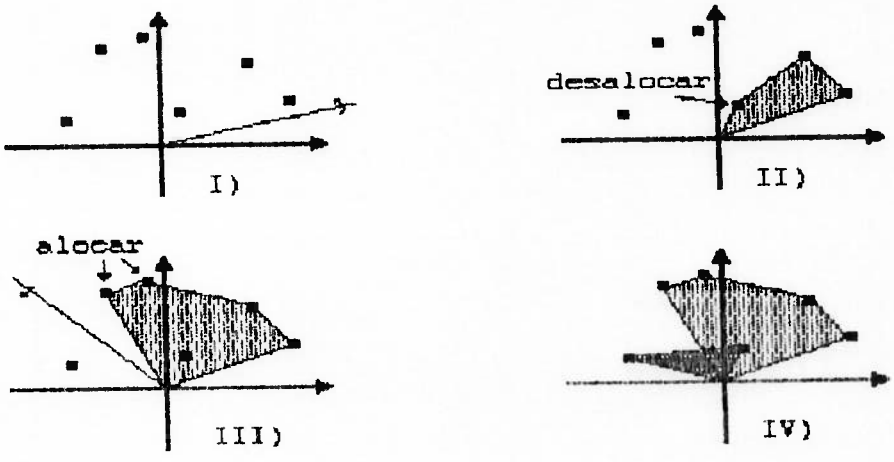
6) Caso a distância máxima admissível seja excedida, retirar os últimos nós dessa viagem, até que essa restrição seja atendida;

7) Substituir um nó dessa viagem por um ou mais nós não alocados, deixando-se o nó substituído para ser alocado em viagens subseqüentes, se ocorrer diminuição da distância total percorrida, e não for violada a restrição de capacidade. O nó i a ser substituído é o que minimiza uma função das coordenadas polares dos nós da viagem, por exemplo, R(i)+A(i).AVR, onde R(i) é o raio, A(i) é o seu ângulo e AVR é o raio médio dos nós dessa viagem. Procura-se a seguir alocar o nó não alocado mais próximo ao último nó alocado na viagem, a seguir o mais próximo ao que se procurou alocar, e assim por diante. Conclue-se o processo quando não for mais possível alocar novos nós a essa viagem.

8) Caso haja nós não alocados a alguma viagem, considerar como nó inicial o primeiro nó da lista não alocado a nenhuma viagem, e repetir os passos 4 a 8;

9) Fim do procedimento.

FIGURA 2.5 – FUNCIONAMENTO DO ALGORITMO DE VARREDURA



A figura 2.5 ilustra o funcionamento da heurística de varredura. I) mostra o problema a ser resolvido. A reta diagonal representa o ângulo

de varredura. Em II) uma viagem completa foi obtida. Em III) essa mesma viagem foi modificada pela substituição de um nó por dois outros. O processo de varredura deve ser repetido a partir do início. Por último IV) representa as duas viagens obtidas como solução final.

Os autores recomendam que, para um dado problema, sejam obtidas diversas soluções, através da variação do critério de ordenação dos nós e do nó inicial da lista, escolhendo-se a solução que apresente menor distância total.

2.5.2- ALGORITMO DE CLARKE & WRIGHT

Publicado em 1962, o algoritmo de Clarke & Wright [9], uma heurística de economia e inserção, é um dos mais utilizados hoje em dia, por ser de execução computacional rápida e por apresentar soluções de boa qualidade. Na sua forma original tem por objetivo minimizar a distância total percorrida, resolvendo problemas de roteamento de múltiplos veículos de capacidades diferentes, partindo de uma única base, sem quaisquer restrições exceto a de capacidade máxima de carga dos veículos.

O funcionamento do algoritmo de Clarke & Wright é o seguinte:

1) Alocar a cada nó um veículo em viagem exclusiva;
2) Calcular, para todos os pares de nós i-j, a economia que resulta na junção dos mesmos em uma única viagem, expressa por $S(i,j) = d_{0I} + d_{0J} - d_{IJ}$.
3) Criar uma lista, ordenando-se decrescentemente as economias;
4) Retirar a primeira economia da lista e obter os nós i e j correspodentes;

5) Se os nós i ou j não forem nós extremos de uma viagem (imediatamente anteriores ou seguintes à base no roteiro), continuar a partir do passo 8;

6) Se a capacidade do veículo não for suficiente para atender à demanda das viagens combinadas, continuar a partir do passo 8;

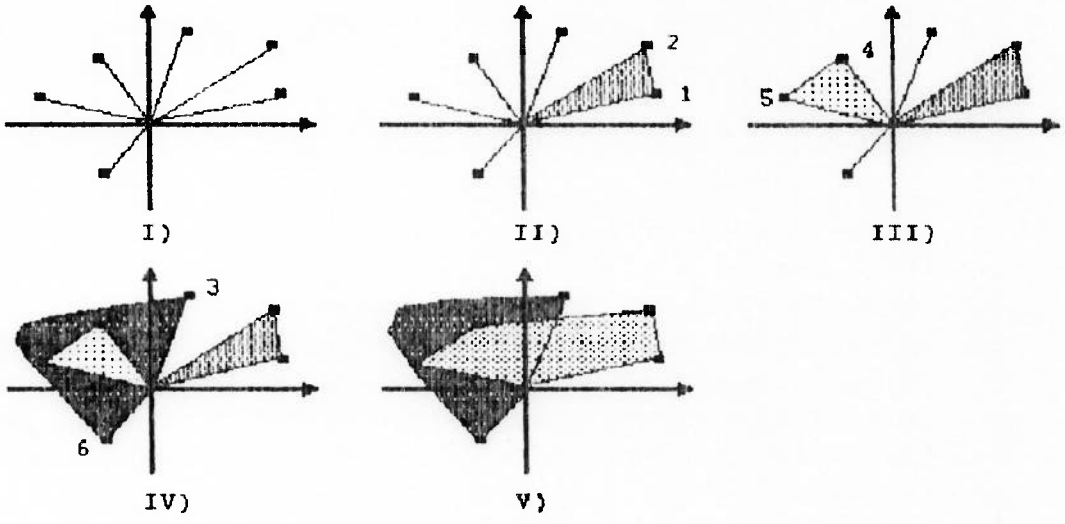
7) Unir os nós i e j através de um arco, o que implica em unir as duas viagens das quais faziam parte os nós i e j em uma única;

8) Caso não tenha sido retirada da lista a última economia, retirar a economia seguinte à atual, obtendo os nós i e j da mesma, e repetir os passos 5 a 8;

9) Aplicando alguma heurística para o problema do caixeiro viajante, otimizar cada viagem formada;

10) Fim do procedimento.

FIGURA 2.6 – FUNCIONAMENTO DO ALGORITMO DE CLARKE & WRIGHT



A figura 2.6 ilustra o funcionamento do algoritmo de Clarke & Wright. I) mostra todos os nós atendidos por veículos em viagens exclusivas. II) a IV) mostram viagens sendo combinadas e V) mostra a solução final, que consiste de duas viagens.

Como um nó, uma vez alocado a uma viagem, não pode ser removido da mesma, Golden et alii [16] recomendam que diversas soluções diferentes sejam geradas pelo algoritmo de Clarke & Wright na resolução de um problema, escolhendo-se a melhor delas como solução final. Isso pode ser conseguido introduzindo-se na formulação de cálculo das economias um termo (μ) que assume qualquer valor (em seu trabalho, Golden et alii [16] variaram μ de 0.0 a 3.0, com incrementos de 0.1), e que, dando um peso maior ou menor à redução de distância conseguida com a combinação de duas viagens, permite a obtenção de soluções diferentes. A formulação para o cálculo das economias passa, então, a ser $S(i,j) = d_{OI} + d_{OJ} - \mu.d_{IJ}$.

2.5.3- ALGORITMO DE PARTIÇÃO DO ROTEIRO GIGANTE

Esse algoritmo é uma heurística de roteamento e partição, proposto por Golden et alii [16], tendo sido desenvolvido para resolver o problema de tamanho e composição de frota, uma extensão do problema de roteamento de múltiplos veículos. O algoritmo considera veículos com capacidades e custos fixos diferentes, e tem como objetivo minimizar o custo total da frota (custo fixo e custo variável), considerando que todos os veículos tem o mesmo custo variável por distância percorrida (Co).

O funcionamento do algoritmo de partição simples do roteiro gigante é o seguinte:

1) Utilizando uma heurística para o problema do caixeiro viajante, criar uma seqüência de nós, que será chamado de roteiro gigante, unindo todos os nós (incluíndo a base). O primeiro ponto desse

roteiro, imposto como sendo a base, é representado como R_0. Seguem-lhe os nós R_1, R_2, ..., R_N no roteiro;

2) Calcular o termo C_{IJ} para todo i e j, $0 \leq i < j \leq N+1$, dado pela equação 2.20, onde $R_0 = R_{N+1} = base$:

$$Cij = \begin{cases} Cf_V + Co.\left(d_{0R(I)} + \sum_{m=i}^{j-2} d_{R(M)R(M+1)} + d_{R(J-1)0} \right) & \text{se } \sum_{m=i}^{j-1} D_{R(M)} \leq Cv \\ \infty \quad \text{se } \sum_{m=i}^{j-1} D_{R(M)} > C_V \end{cases}$$

, para $v = 1, \ldots, NTV$ [2.20]

Esse termo representa o custo total de, partindo-se da base, percorrer-se o trecho do roteiro gigante do $i_{ésimo}$ ao $(j-1)_{ésimo}$ nó, retornando a seguir à base. O veículo utilizado será o de menor capacidade que puder realizar a viagem sem violar a restrição de capacidade da equação 2.21, para $1 < v < NTV$ (o número de tipos diferentes de veículos disponíveis).

$$\sum_{m=i}^{j-1} D_{R(m)} \leq C_V \qquad [2.21]$$

Se nenhum tipo de veículo conseguir realizar a viagem sem violar a restrição de capacidade, então C_{IJ} será infinito;

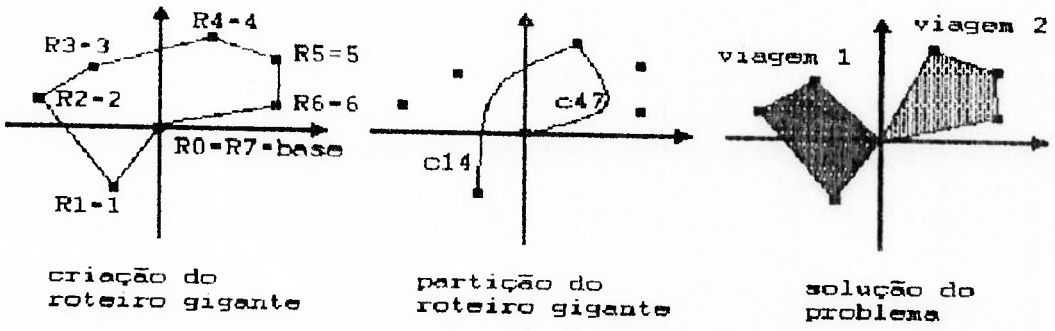
3) Criar uma rede orientada de nós, onde cada nó i está conectado ao nó j, $1 \leq i < j \leq N+1$, através de um arco de custo C_{IJ}. Cada nó i dessa rede orientada representa o nó R_I do roteiro gigante, $1 \leq i \leq N$;

4) Determinar o caminho de menor custo entre os nós 1 e N+1 da rede orientada. Os nós que fizerem parte do caminho mínimo serão os nós

nos quais será iniciada uma nova partição do roteiro gigante (uma nova viagem). O custo desse caminho é o custo total da frota.

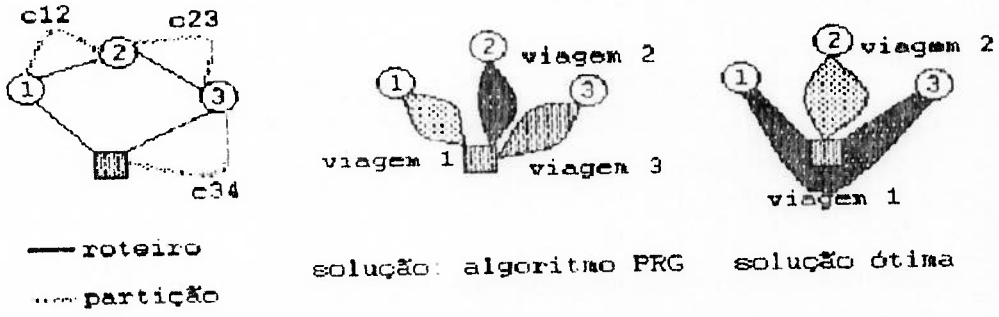
A figura 2.7 ilustra o funcionamento do algoritmo de partição do roteiro gigante. Em I) é apresentado o roteiro gigante que une os seis nós do problema, em II) o caminho de menor custo, que passa pelos nós assinalados por setas, e em III), a solução final, que consiste de duas viagens.

FIGURA 2.7 – FUNCIONAMENTO DO ALGORITMO DE PARTIÇÃO DO ROTEIRO GIGANTE



O algoritmo de partição do roteiro gigante apresenta o inconveniente de seguir rígidamente a seqüência de nós do roteiro gigante, o que pode levar a casos patológicos como o ilustrado na figura 2.8.

FIGURA 2.8 - CASO PATOLÓGICO: ALGORITMO DE PARTIÇÃO DO ROTEIRO GIGANTE



Obs: D1=5, D2=10, D3=5
Capacidade dos veículos=10

Para reduzir a probabilidade da ocorrência de casos como esses, foi desenvolvido o algoritmo de partição múltipla do roteiro gigante. A idéia do mesmo é variar o primeiro nó do roteiro gigante, no qual se iniciará a primeira viagem, visando a obtenção de soluções diferentes para um mesmo problema. Golden et alii [16] propõe, após a obtenção de uma partição do roteiro gigante, iniciar o processo de partição do roteiro gigante pelo nó seguinte ao pelo qual foi iniciado o processo anterior de partição, ou, em outras palavras, fazer R(1)'=R(2), R(2)'=R(3), ... , R(N-1)'=R(N), R(N)'=R(1). O procedimento deve ser repetido M vezes, sendo M obtido pela equação 2.22.

$$\begin{cases} \sum_{i=1}^{M-1} D_{R(I)} < C_V \leq \sum_{i=1}^{M} D_{R(I)} \\ \\ C_V = \max [C_1\ C_2\ \ldots\ C_{NTV}] \end{cases} \qquad [2.22]$$

A solução final é a que apresentar menor custo dentre todas as M soluções obtidas.

2.5.4- ALGORITMO DE FISHER & JAIKUMAR

O algoritmo de Fisher & Jaikumar [11] é uma heurística baseada em programação matemática, que resolve problemas de roteamento de múltiplos veículos com capacidades diferentes.

Fisher & Jaikumar partem da formulação matemática para o problema de roteamento de múltiplos veículos de capacidades variáveis expressa nas equações [2.23] a [2.30].

$$\text{minimizar} \sum_{ijk} c_{ij} \cdot x_{ijk} \qquad [2.23]$$

sujeito a

$$\sum_{i} D_i \cdot Y_{ik} \leq C_k \qquad k=1 \,..\, NV \qquad [2.24]$$

$$\sum_{k} Y_{ik} = \begin{cases} K \text{ se } i=0 \\ 1 \text{ se } i = 1 \,..\, N \end{cases} \qquad [2.25]$$

$$Y_{ik} = 0 \text{ ou } 1 \qquad \begin{matrix} i=0 \,..\, N \\ k=1 \,..\, NV \end{matrix} \qquad [2.26]$$

$$\sum_{i} x_{ijk} = Y_{jk} \qquad \begin{matrix} j=0 \,..\, N \\ k=1 \,..\, NV \end{matrix} \qquad [2.27]$$

$$\sum_{j} x_{ijk} = Y_{ik} \qquad \begin{matrix} i=0 \,..\, N \\ k=1 \,..\, NV \end{matrix} \qquad [2.28]$$

$$\sum_{ij \in S \times S} x_{ijk} \leq |S| - 1, \qquad \begin{matrix} S \subseteq (1 \,..\, n) \\ 2 \leq |S| \leq n-1 \end{matrix} \qquad k=1 \,..\, NV \qquad [2.29]$$

$$x_{ijk} = 0 \text{ ou } 1 \qquad \begin{matrix} i=0 \,..\, N \\ j=0 \,..\, N \end{matrix} \qquad [2.30]$$

, onde y_{IK} é 1, se o nó i é atendido pelo veículo k, 0 em caso contrário; x_{IJK} é 1 se o veículo k sai do nó i tendo como destino o nó j; c_{IJ} é o custo de, saindo-se do nó i, dirigir-se ao nó k.

A função objetivo 2.23 visa a minimização do custo de viagens do problema. A equação 2.24 tem por objetivo evitar que uma viagem exceda a capacidade do veículo que a atenda. As equações 2.25 e 2.26 garantem que cada viagem comece e termine na base e que todo nó seja atendido por um veículo. As equações 2.27 a 2.30 definem, para cada veículo k, um problema de caixeiro viajante para os nós alocados a esse veículo. Escrevendo-se o problema de roteamento de múltiplos veículos dessa forma, pode-se observar que o mesmo consiste de dois problemas interrelacionados, um que agrupa os nós (chamado por Fisher & Jaikumar de problema generalizado de alocação), e um que roteiriza os nós agrupados a cada veículo k (problema de caixeiro viajante).

O algoritmo de Fisher & Jaikumar resolve inicialmente o problema da alocação dos nós aos veículos minimizando uma função objetivo que aproxima o custo dos roteiros-caixeiro-viajante feitos por cada veículo para atender os nós que lhe são alocados, não levando em conta a posição relativa de um nó dentro de cada grupo. Isso leva à necessidade de, feita a alocação dos nós em grupos, determinar a seqüência dos nós em cada grupo, através de uma resolução, exata ou heurística, do problema de caixeiro viajante.

O problema de roteamento de múltiplos veículos das equações 2.23 a 2.30 pode ser reformulado como um problema não-linear generalizado de alocação, como nas equações 2.31 a 2.34,

$$\text{minimizar} \sum_{k} f(y_k) \qquad [2.31]$$

sujeito a

$$\sum_i D_i \cdot y_{ik} \leq C_k \qquad k=1 \,..\, NV \qquad [2.32]$$

$$\sum_k y_{ik} = \begin{cases} K \text{ se } i=0 \\ 1 \text{ se } i = 1 \,..\, N \end{cases} \qquad [2.33]$$

$$y_{ik} = 0 \text{ ou } 1 \qquad \begin{matrix} i=0 \,..\, N \\ k=1 \,..\, NV \end{matrix} \qquad [2.34]$$

onde a função objetivo $f(y_K)$, o custo de um roteiro-caixeiro-viajante ótimo para os nós em $N(y_K)=\{i \mid y_{ik}=1\}$, pode ser definida matematicamente pelas equações 2.35 a 2.39.

$$f(y_k) = \sum_{ijk} c_{ij} \cdot x_{ijk} \qquad [2.35]$$

sujeito a

$$\sum_i x_{ijk} = y_{jk} \qquad j=0 \,..\, N \qquad [2.36]$$

$$\sum_j x_{ijk} = y_{ik} \qquad i=0 \,..\, N \qquad [2.37]$$

$$\sum_{ij \in S \times S} x_{ijk} \leq |S| - 1, \qquad \begin{matrix} S \subseteq \{1 \,..\, n\} \\ 2 \leq |S| \leq n-1 \end{matrix} \qquad [2.38]$$

$$x_{ijk} = 0 \text{ ou } 1 \qquad \begin{matrix} i=0 \,..\, N \\ j=0 \,..\, N \end{matrix} \qquad [2.39]$$

Sendo a função objetivo muito complexa, Fisher & Jaikumar obtiveram uma aproximação linear para $f(y_k)$, $\sum_{i=1}^{n} a_{ik} \cdot y_{ik}$, e resolvendo o problema apresentado nas equações 2.31 a 2.34 substituindo a equação 2.31 pela equação 2.40,

Na implementação computacional realizada pelos autores, utilizou-se o algoritmo de Miliotis, que consiste numa combinação de "branch & bound" com planos de corte [4], utilizando-se relaxação lagrangeana para a solução do problema de alocação, na qual os multiplicadores são determinados por um método de ajuste de multiplicadores, descrito por Fisher [10].

O algoritmo de Fisher & Jaikumar original não prevê restrições de duração de jornada, mas os autores, para realizar os testes comparativos, descritos em [11], utilizaram uma aproximação linear para a duração do roteiro em cada grupo de nós, utilizando a mesma em uma restrição aproximada de duração de jornada, adicionada às equações 2.40 e 2.32 a 2.34. Convém ressaltar que essa restrição é aproximada, sendo necessária a verificação final de que realmente a duração máxima de jornada não foi excedida por nenhum dos roteiros obtidos.

2.6- MÉTODOS HEURÍSTICOS PARA O PROBLEMA "DIAL-A-RIDE"

Problemas "dial-a-ride" são aqueles nos quais um cliente solicita a uma companhia que opera veículos para levá-lo de uma origem a um destino determinados, que não necessáriamente são a base da frota, devendo-se determinar qual dos veículos da frota deverá atender a esse cliente, e em que posição do seu roteiro deverá ser feita a coleta e a entrega. Problemas dial-a-ride com janelas de tempo incluem a necessidade da coleta e da entrega serem feitas em intervalos de tempo determinados, por exemplo, coleta entre 7:00 e 7:30 e entrega entre 11:00 e 12:30 horas.

O problema "dial-a-ride" difere do problema objetivo da presente dissertação, porém um dos diversos algoritmos existentes, o algoritmo ADARTW [19] (Algorithm Dial-A-Ride with Time Windows) será apresentado

na revisão bibliográfica pois alguns dos seus princípios de funcionamento foram utilizados no desenvolvimento dos algoritmos propostos na presente dissertação.

2.6.1- ALGORITMO ADARTW

O algoritmo para problemas dial-a-ride com janelas de tempo ADARTW, desenvolvido por Jaw et alii [19], resolve problemas nos quais diversas cargas tem que ser recolhidos em pontos de origem e entregues em pontos de destino em horários determinados, dispondo-se para tanto de uma frota de veículos. A capacidade dos veículos é considerada ilimitada, a carga e descarga dos produtos é considerada instantânea, e cada carga não pode permanecer no veículo por um tempo superior ao especificado.

O princípio do algoritmo, que é de interesse na presente dissertação, é o de composição de viagens, ou seja, um dado produto, é alocado a um dado veículo em uma dada posição do roteiro, se a sua inclusão não acarretar a violação de nenhum intervalo de coleta e entrega dessa carga e de nenhuma das cargas já alocadas a esse veículo. O algoritmo é bastante míope, não considerando o efeito que a alocação de uma carga a um veículo trará nos demais veículos e na qualidade da solução final obtida.

O funcionamento do algoritmo ADARTW, em linhas gerais, é o seguinte:

1) Ordenar todas as cargas segundo, por exemplo, horários iniciais de coleta, obtendo uma lista;
2) Retirar a primeira carga da lista;
3) Para todos os veículos, verificar qual a posição de inserção da carga (ponto de coleta e ponto de entrega) no roteiro, se houver,

que não acarrete a violação dos intervalos de coleta e entrega da mesma e de todas as demais cargas alocadas a esse veículo, e que resulte em um acréscimo de custo mínimo;

4) Alocar a carga ao veículo que resulte em um menor acréscimo de custo;

5) Se houver cargas não alocadas a nenhum veículo, retirar a próxima carga da lista e executar os passos 3,4 e 5;

6) Fim do algoritmo;

A descrição não entrou em detalhes, como o procedimento para a determinaçao da viabilidade de uma inserção dos pontos de coleta e de entrega de um produto em um roteiro ou o detalhamento do custo de inserção, que são apresentados no trabalho original [19].

O procedimento, apesar de míope, é extremamente flexível, permitindo incorporar práticamente qualquer restrição encontrada em um problema real.

2.7- CRITÉRIOS PARA A COMPARAÇÃO DA EFICIÊNCIA DE HEURÍSTICAS

São relevantes à presente dissertação os seguintes critérios para a comparação de heuristicas:

- Flexibilidade da heurística;
- Qualidade da solução obtida;
- Tempo de processamento;

Algumas heurísticas, apesar de desenvolvidas para um problema específico, podem ser fácilmente adaptadas para incorporar restriçoes e características adicionais, como, por exemplo, o algoritmo de partição do roteiro gigante ou o algoritmo ADARTW. Diz-se, então, que essas heurísticas são mais flexíveis que outras heurísticas, sendo essa propriedade que determinou que esses dois algoritmos fossem utilizados como base para o desenvolvimento dos algoritmos para dimensionamento de frota e para o roteamento de uma frota existente, respectivamente, na presente dissertação.

A qualidade da solução obtida com a heurística é importante, pois quanto mais próxima da solução ótima for a solução obtida pela heurística, menores serão os gastos reais com o transporte. Cabe observar que uma dada heuristica pode apresentar um comportamento muito bom em um dado problema e um desempenho medíocre em um outro problema, se o mesmo for um caso patológico. Dessa forma, além de se conhecer o desvio médio entre a solução obtida e a solução ótima, é conveniente conhecer-se também o seu desvio padrão, pois não é aconselhável a utilização de heurísticas que podem obter tanto soluções extremamente próximas quanto muito distantes da ótima.

Em algoritmos para roteamento de múltiplos veículos é extremamente difícil comparar-se uma solução obtida com a soluçao ótima pela dificuldade na obtenção desta. Por isso é mais comum comparar-se soluções obtidas por diversos algoritmos, obtendo-se, dessa forma, não o quão boa é uma solução, mas o quanto melhor a mesma é quando comparada com a obtida por outros algoritmos. A tabela 2.1 é uma transcrição de uma comparação feita por Fisher & Jaikumar [11] entre os algoritmos de Clarke & Wright, de Varredura e de Fisher & Jaikumar em 12 problemas diferentes. Na tabela 2.1, a distância (Dist.) é dada em milhas e o tempo de processamento (T.CPU) em segundos.

TABELA 2.1- COMPARAÇÃO DE DESEMPENHO DE TRÊS ALGORITMOS

Problema	Nós	C & W		G & M		F & J	
		Dist.	T.CPU	Dist.	T.CPU	Dist.	T.CPU
1	50	585	0.80	532	12.20	524	1.33
6	50	619	1.30	560	11.40	560	2.17
2	75	900	1.70	874	24.30	857	1.71
7	75	976	2.60	933	23.80	916	2.94
3	100	886	2.40	851	65.10	833	2.53
8	100	973	3.50	888	58.50	885	7.46
11	100	831	2.40	937	50.80	824	0.87
12	100	877	2.90	949	53.60	848	1.94
4	150	1204	6.60	1079	142.00	1014	4.80
9	150	1426	11.20	1230	134.70	1230	17.33
5	199	1540	11.00	1389	252.20	1420	5.73
10	199	1800	14.00	1518	238.50	1518	19.51
Média		978.3	88.93	1051.4	5.03	952.4	5.69

Outra característica de uma heurística é o seu tempo de processamento. Conhecendo-se a duração média de um algoritmo, parametrizada com características do problema, pode-se determinar qual o ambiente em que a implementação computacional deve ser feita (computador de grande porte, microcomputador, ou outros).

3- DESCRIÇÃO DO CENÁRIO PADRÃO ADOTADO

3.1- INTRODUÇÃO

Os algoritmos existentes de roteamento de múltiplos veículos possuem características que são adequadas aos problemas para os quais foram originàriamente desenvolvidos, sendo que, na maioria das vezes tornam-se necessárias modificações para a resolução de problemas com características diferentes do problema original.

Essas alterações, quando possíveis, são de difícil implementação e, freqüentemente, acabam gerando algoritmos que diferem significativamente dos originais.

Na presente dissertação foi utilizado um cenário abrangente, do qual muitos cenários reais sejam simplificações, podendo os algoritmos desenvolvidos na presente dissertação ser utilizados sem alterações nesses cenários simplificados.

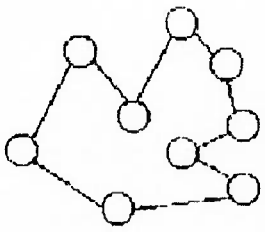
Neste capítulo será apresentado o cenário padrão adotado na presente dissertação. O cenário do problema do caixeiro viajante será desenvolvido até obter-se o cenário padrão, que será detalhado no final do capítulo.

3.2- OBTENÇÃO DO CENÁRIO PADRÃO

Como apresentado no capítulo 2, o problema do caixeiro viajante consiste em percorrer-se um conjunto de cidades (ou pontos de demanda),

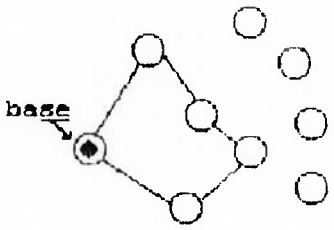
cada qual uma única vez, de forma que a distância total percorrida seja mínima. O resultado é um roteiro como apresentado na figura 3.1.

FIG 3.1 - CAIXEIRO VIAJANTE



Se o número de pontos de demanda for muito grande, ou os mesmos forem muito distantes entre si, pode ocorrer que o tempo gasto para percorrer todo o roteiro seja muito alto, o que, em problemas reais, pode ser indesejável. O problema do caixeiro viajante com restrição de duração, um desenvolvimento do problema do caixeiro viajante, tem por objetivo obter um sub-roteiro (que não necessáriamente inclua todos os pontos de demanda), que maximize o lucro obtido com a viagem, iniciando-se e terminando-se a mesma em um ponto de demanda base, de forma que não se exceda uma duração máxima especificada.

FIG 3.2 - CAIXEIRO VIAJANTE COM RESTRIÇÃO DE DURAÇÃO



É necessário, para tanto, que se conheça a duração da viagem entre cada dois pontos de demanda do problema, bem como o lucro esperado com essa viagem. O resultado de um problema de caixeiro viajante com restrição de duração é apresentado na figura 3.2.

O problema do caixeiro viajante com restrição de capacidade é análogo ao do caixeiro viajante com restrição de duração, porém o fator limitante da viagem é a capacidade de carga do veículo, deixando-se

pontos de demanda sem atendimento devido à incapacidade de embarcar-se a carga no veículo.

Se nenhum ponto de demanda puder ficar sem ser atendido, (não podendo se exceder um período de tempo especificado, ou se a capacidade do veículo for insuficiente para atender a todos os pontos de demanda em uma única viagem), mais de um veículo são necessários para atender o problema.

No problema básico de roteamento de múltiplos veículos, como apresentado no capítulo 2, um conjunto de pontos de demanda deve ser atendido por uma frota de veículos, de capacidades conhecidas, com as viagens sendo limitadas pela capacidade.

- Um conjunto de pontos de demanda, sendo a demanda (do único tipo de carga) de cada um conhecida e não necessariamente igual à dos demais;
- Uma única base na qual se iniciam e terminam todas as viagens;
- Cada viagem realizada por um único veículo de uma frota, com capacidade máxima de carga conhecida;
- A frota possui tantos veículos quantos forem necessários para que nenhum ponto de demanda deixe de ser atendido;
- O objetivo é atender a todos os pontos de demanda percorrendo-se a menor distância total possível;

O problema básico de roteamento de múltiplos veículos é um problema bastante estudado, com diversos trabalhos publicados abordando o tema, e com diversas metodologias desenvolvidas para resolvê-lo (algumas das quais foram apresentadas no capítulo 2. Muitos problemas de transporte reais são problemas com características do problema básico de roteamento

de múltiplos veículos (como exemplo pode-se citar o "Problema de transporte de pessoal para as plataformas da bacia de Campos" [3], um problema básico de roteamento de múltiplos veículos com restrição de duração). Entretanto, outros problemas podem possuir características que difiram do problema básico, apesar de serem problemas de roteamento de múltiplos veículos, o que pode dificultar a sua resolução com as metodologias desenvolvidas para o problema básico.

Golden et alii definem o problema de tamanho e composição de frota como sendo o problema de roteamento de múltiplos veículos no qual o enfoque é na obtenção da solução de menor custo, e não de menor distância percorrida, sendo que a diferentes tipos de veículos estão associados diferentes custos. Dessa forma, mais do que rotear uma frota de veículos fornecida, trata-se de definir uma frota, a partir de um conjunto de opções, de forma que, não apenas a distância percorrida seja mínima, mas também o custo da frota seja mínimo, um problema que ocorre sempre que não se dispõe de uma frota para atender os pontos de demanda, ou quando deseja-se renovar a frota, passando-se a considerar custos.

Nessa seção, partindo-se do problema do caixeiro viajante, através da inclusão de novas características, definiu-se problemas encontrados com freqüência na prática. A seguir será definido o problema geral de roteamento de múltiplos veículos, objeto de estudo da presente dissertação.

3.3- O CENÁRIO PADRÃO DO PROBLEMA GERAL DE ROTEAMENTO DE MÚLTIPLOS VEÍCULOS

O cenário padrão do problema geral de roteamento de múltiplos veículos, adotado na presente dissertação, continua tendo as

características do problema básico de roteamento de múltiplos veículos, ao qual foram adicionadas as seguintes características adicionais:

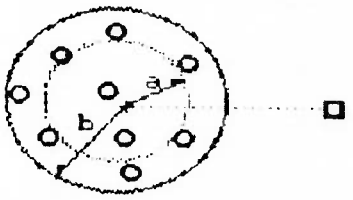
- É imposta uma duração máxima de jornada, que nunca deve ser excedida;
- A duração do embarque na base e do desembarque nos pontos de demanda é função da quantidade de carga envolvida e da razão de carregamento dos equipamentos de manuseio de carga em cada ponto;
- A duração do translado entre dois pontos quaisquer (inclusive a base) é função da distância entre os mesmos e da velocidade do veículo utilizado;
- É permitida a cada veículo da frota a realização de mais de uma viagem, desde que a soma da duração total das viagens (translado, carga e descarga) não exceda a duração de jornada especificada;
- É admitida a possibilidade de um ponto de demanda ser atendido por mais de um veículo.

O cenário padrão pode ser representado esquematicamente pela Figura 3.3, que decompõe o mesmo em dois grupos distintos: a base e os pontos de demanda. Através dessa modelagem permite-se obter problemas geométricamente distintos variando-se os seguintes parâmetros:

- Número de pontos de demanda (N);
- Distância média dos pontos de demanda ao seu centro geométrico (μd_+);
- Distância da base ao centro geométrico dos pontos de demanda (μd_0);
- Demanda dos pontos de demanda (D);
- Taxa de carregamento nos pontos de demanda e na base (TC);

- Duração da jornada (TV);

FIGURA 3.3- CENÁRIO PADRÃO



a) μd_o — base

b) $\mu d_o \sqrt{2}$ — centro de demanda

Para a realização de testes de desempenho e comparativos dos algoritmos desenvolvidos na presente dissertação, criou-se um programa, descrito no apêndice, que gera problemas aleatórios com uma geometria definida. Dessa forma, foi possível analisar a influência que cada componente do cenário tem sobre o comportamento dos algoritmos e das soluções obtidas.

3.4- A FROTA DE VEÍCULOS

Para atender à demanda de transporte do cenário deve ser utilizada uma frota de veículos com as seguintes características:

- Existe um número conhecido de tipos diferentes de veículos que podem compor a frota;
- Cada tipo é caracterizado por uma dada velocidade de serviço e uma dada capacidade de transporte de carga (peso ou volume);
- A cada tipo de veículo está associado um custo fixo;
- A cada tipo de veículo está associado um custo variável unitário, sendo o custo variável total do veículo função da distância;

A disponibilidade ou não de infinitos veículos de cada tipo para atender os pontos de demanda determinou a formulação de dois problemas diferentes, o de dimensionamento de frota e o de alocação de frota.

3.5- OS PROBLEMAS GERAIS DE DIMENSIONAMENTO E DE ALOCAÇÃO DE FROTA

O objetivo visado na presente dissertação de mestrado não é a minimização da distância total percorrida, como no problema básico de roteamento de múltiplos veículos iguais, mas sim a minimização do custo total da frota.

O custo da frota é composto por uma parcela fixa, que não depende da operação da mesma, e por uma parcela variável, que é nula no caso de nenhuma viagem ser realizada.

A parcela fixa, chamada custo fixo total, depende do número e dos tipos dos veículos que compõem a frota, como o custo de aquisição dos veículos, administração da frota, salários de mensalistas, e equivale à soma dos custos fixos de todos os veículos da frota.

A parcela variável, chamada de custo variável total, dependente da distância percorrida, é tanto maior quanto maior for o número de viagens realizadas pela frota. É devida a gastos que somente ocorrem com a utilização da frota, como consumo de combustíveis e lubrificantes e gastos com manutenção, entre outros, e equivale à soma dos custos operacionais de todos os veículos da frota.

Se o objetivo do problema geral de roteamento de múltiplos veículos for dimensionar uma frota de veículos para atender a um conjunto de

pontos de demanda, minimizando o custo total da frota, tem-se um problema de dimensionamento de frota. Se já houver uma frota de veículos, e o objetivo do problema for alocar eficientemente a mesma, minimizando o seu custo variável, tem-se um problema de alocação de frota.

3.6- CONSIDERAÇÕES FINAIS

O problema geral de roteamento de múltiplos veículos apresentado neste capítulo, apesar de bastante abrangente, ainda possui simplificações, que restringem a aplicação das metodologias desenvolvidas na presente dissertação em problemas reais.

É imposta uma unica base, de localização e caraterísticas de manuseio de carga conhecidas. Devido a isso, as metodologias desenvolvidas não tem capacidade de determinar a localização de bases dentro de um conjunto de pontos de demanda, ou, no caso do problema a ser resolvido possuir mais de uma base, distribuir a frota entre as mesmas, e considerar a possibilidade de viagens que comecem e terminem em bases diferentes.

Também é imposta uma duração de jornada para todos os veículos da frota, não sendo considerada a possibilidade de diferentes veículos (ou tipos de veículos) da frota terem duração máxima de operação diferentes.

Por último, não se considera a possibilidade do transporte de mais de um tipo de carga, e a eventual interferência dos tipos de carga, ou de capacidade de transporte simultâneo de diferentes tipos de carga diferentes para os tipos de veículos de carga.

4- ALGORITMO PARA O PROBLEMA DE DIMENSIONAMENTO DE FROTA

4.1- INTRODUÇÃO

O algoritmo de partição múltipla do roteiro gigante (Golden et alii [16]) tem o enfoque de minimizar custos, visando determinar quantos veículos, e de que capacidade, devem ser utilizados para atender a demanda a um mínimo custo. Esse problema, batizado pelos autores de problema de tamanho e composição de frota (fleet size and mix problem), deve também otimizar a composição da frota, além de seu tamanho.

O problema geral estudado na presente dissertação é um problema de tamanho e composição de frota, diferindo em quatro pontos, que impedem a utilização direta do algoritmo de partição de roteiro gigante:

- É permitida a ocorrência de pontos de demanda com demanda superior à capacidade do maior tipo de veículo disponível, o que gera a necessidade de um ponto de demanda ser atendido por mais de um veículo;
- É permitido que um veículo da frota realize mais de uma viagem. Com isso não necessáriamente o veículo alocado a uma viagem será o que a realize com o menor custo, mas o que resultará no menor acréscimo de custo total (fixo e variável) atendendo as viagens alocadas a ele.
- Os tipos de veículos podem ter, além de custos fixos e capacidades diferentes, custos variáveis e velocidades diferentes;
- Existe uma restrição de duração máxima de jornada, que limita o número de viagens que um veículo pode realizar, e que, conseqüentemente, influe na composição da frota.

As duas últimas diferenças entre os problemas podem ser fácilmente incorporadas ao algoritmo de partição múltipla de roteiro gigante, como realizado por Brinati et alii [3], substituíndo-se a equação 2.20, para cálculo dos custos C_{IJ} das possíveis partições do roteiro gigante em viagens, pelas equações 4.1 a 4.3, utilizando-se para atender a viagem do i-ésimo ao j-ésimo nó do roteiro gigante o veículo v ($1 \leq v \leq NTV$) que,

$$\text{minimizar } Cij = Cf_v + Co_v \cdot \left(d_{0R(i)} + \left(\sum_{m=i}^{j-2} d_{R(m)R(m+1)} \right) + d_{R(j-1)0} \right) \qquad [4.1]$$

sujeito a

$$\sum_{m=i}^{j-1} D_{R(m)} \leq C_v \qquad [4.2]$$

$$\sum_{m=i}^{j-1} D_{R(m)} \cdot \left(\frac{Tc_{R(m)} + Tc_0}{Tc_{R(m)} \cdot Tc_0} \right) + \frac{d_{0R(i)} + \sum_{m=i}^{j-2} d_{R(m)R(m+1)} + d_{R(j-1)0}}{Vs_v} \leq TV \qquad [4.3]$$

sendo que $C_{IJ} = \infty$ se nenhum tipo de veículo puder atender a viagem sem violar as restrições 4.2 e 4.3.

Comparando-se com a equação 2.20, observa-se que o custo variável por unidade de distância Co passa a depender do tipo de veículo, tornando-se Co_v na equação 4.1. Além disso foi introduzida uma restrição de duração de jornada, equação 4.3, que calcula a duração total da viagem, (duração total de transbordo mais duração total de deslocamento do veículo), que deve ser menor ou igual que a duração máxima de jornada. Com isso, o custo fixo Cf_v é o custo fixo do veículo de tipo v no período de tempo correspodente à duração máxima de jornada.

As duas primeiras diferenças mencionadas entre o problema de tamanho e composição de frota e o problema geral de roteamento de múltiplos veículos, entretanto, requerem alterações maiores no algoritmo de partição múltipla de roteiro gigante, que resultaram no algoritmo de dimensionamento de frota, apresentado no presente capítulo.

4.2- ALGORITMO PARA O PROBLEMA DE DIMENSIONAMENTO DE FROTA

O diagrama de blocos da figura 4.1 ilustra o funcionamento do algoritmo de partição múltipla de roteiro gigante adaptado para o problema de dimensionamento de frota, doravante chamado de algoritmo de dimensionamento de frota.

Passo 1: Decomposição os pontos de demanda com demanda superior à capacidade do maior modelo de veículo disponível em nós fictícios;

Passo 2: Criação do roteiro gigante;

Passo 3: Determinação do número M de partições do roteiro gigante a serem realizadas;

Passo 4: Se foram realizadas M partições do roteiro gigante, processar a partir do passo 8, senão processar a partir do passo 5;

Passo 5: Criação de uma rede orientada, utilizando o fator redutor de custo fixo no cálculo de C_{IJ};

Passo 6: Partição do roteiro gigante, determinando-se na rede o caminho de mínimo custo e obtendo-se as viagens;

Passo 7: Deslocamento da posição da base no roteiro gigante, tornando imediatamente anterior à mesma o nó imediatamente posterior. Retorno ao passo 4;

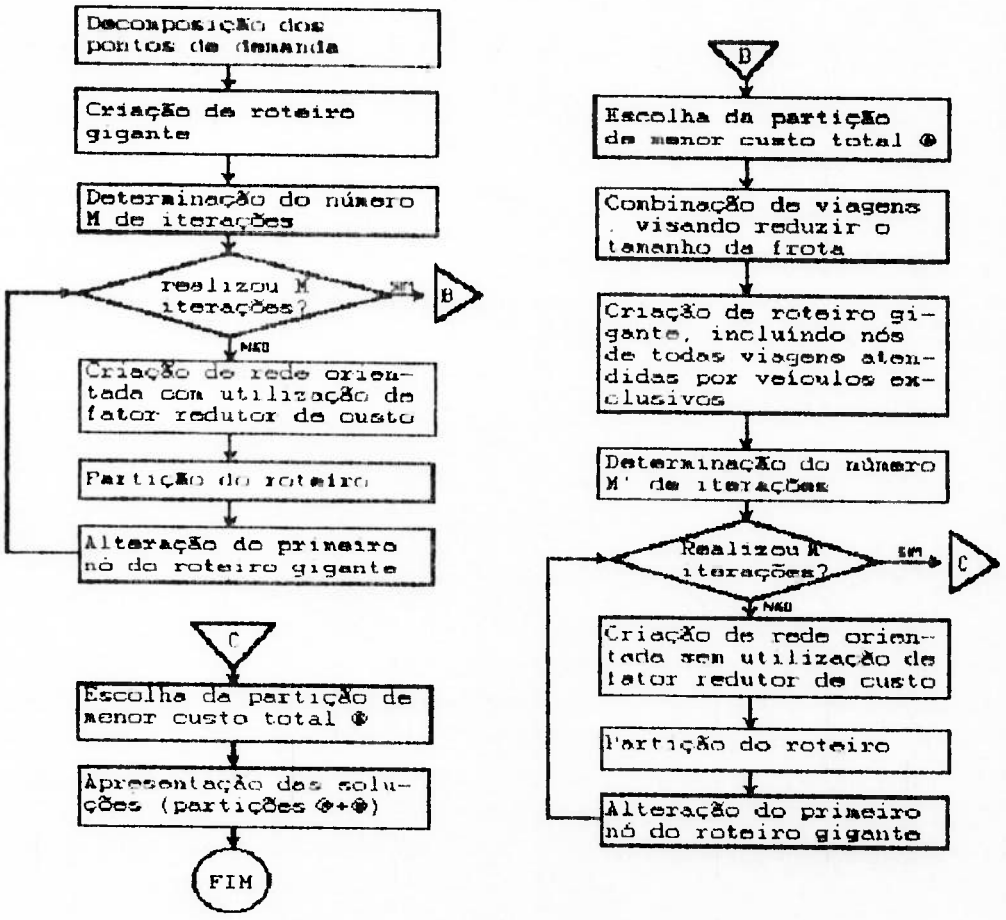


FIGURA 4.1 - ALGORITMO DE DIMENSIONAMENTO DE FROTA

Passo 8: Escolher das M soluções a de menor custo;

Passo 9: Combinação de viagens visando a realização de duas ou mais viagens por um único veículo, quando possível;

Passo 10: Criação de novo roteiro gigante que inclua apenas os nós que façam parte de viagens que sejam atendidas por veículos exclusivos, independentemente de ter ou não sido utilizado o fator redutor de custo fixo nas mesmas;

Passo 11: Determinação do número M' de partições a serem realizadas no novo roteiro gigante;

Passo 12: Se todas as M' partições foram realizadas, processar a partir do passo 16, senão processar a partir do passo 13;

Passo 13: Criação de nova rede orientada, porém com os custos C_{IJ}' obtidos sem a utilização do fator redutor de custo fixo;

Passo 14: Partição do roteiro gigante, determinando-se na rede o caminho de mínimo custo e obtendo-se novas viagens;

Passo 15: Deslocamento da posição da base no roteiro gigante, tornando imediatamente anterior à mesma o nó imediatamente posterior, e processar a partir do passo 12;

Passo 16: Escolha da solução de menor custo dentre as M' soluções obtidas, sendo essa a solução final obtida pelo algoritmo de dimensionamento de frota.

4.3- DETALHAMENTO DOS PRINCIPAIS PASSOS DO ALGORITMO

Serão detalhados a seguir os principais passos do algoritmo.

➲ DECOMPOSIÇÃO DE PONTOS COM DEMANDA SUPERIOR À CAPACIDADE DO MAIOR MODELO DE VEÍCULO DISPONÍVEL

O cenário apresentado no capítulo 3 permite a possibilidade de um ponto de demanda ser atendido por mais de um veículo. Isso é necessário quando a demanda de um ponto é tão elevada que nenhum dos modelos de veículos disponíveis possui capacidade suficiente para atendê-la em uma única viagem.

O primeiro passo do algoritmo consiste na decomposição dos pontos com excesso de demanda em tantos pontos quantos forem necessários, de forma que cada um deles tenha demanda tal que possa ser atendido por pelo

menos um dos tipos disponíveis. Cada um desses pontos de demanda fictícios possui a mesma localização e equipamentos de descarga que o ponto de demanda original, que, por sua vez, tem uma demanda residual igual à demanda original, subtraída da demanda dos pontos de demanda fictícios criados.

A determinação do número de pontos fictícios para cada pontos com excesso de demanda e a demanda dos mesmos será feita determinando-se qual o modelo de veículo que conseguirá minimizar o custo total dos veículos alocados aos pontos fictícios gerados, como expresso nas equações 4.4 a 4.6, onde NF_{IV} é o número de pontos de demanda fictícios do i-ésimo ponto de demanda do roteiro gigante a serem criados utilizando-se um veículo do tipo v.

$$\text{minimizar } NF_{iv} \cdot (F.Cf_v + 2.d_{0R(i)}.Co_v) \qquad [4.4]$$

sujeito a

$$\frac{2.d_{0R(i)}}{Vs_v} + C_v \cdot (Tc_{R(i)} + Tc_0) \leq TV \qquad [4.5]$$

$$NF_{iv} = \begin{cases} \text{trunc}\left(\frac{D_{R(i)}}{C_v} - 1\right) & \text{se resto}\left(\frac{D_{R(i)}}{C_v}\right) = 0 \\ \text{trunc}\left(\frac{D_{R(i)}}{C_v}\right) & \text{se resto}\left(\frac{D_{R(i)}}{C_v}\right) \neq 0 \end{cases} \qquad [4.6]$$

O termo F é o fator redutor de custo fixo, a ser apresentado adiante neste capítulo. Trunc[x] é o maior inteiro menor ou igual a x.

Para cada ponto i com excesso de demanda teremos, então, NF_{iv} pontos fictícios, cada qual com demanda C_v. O ponto i original passará a ter demanda $[D_{R(i)} - NF_{iv} \cdot C_v]$.

O número total N de pontos de demanda será igual ao número original de pontos de demanda, somado ao número total de nós fictícios criados.

➲ CRIAÇÃO DO ROTEIRO GIGANTE

Seguindo o conceito do algoritmo de partição do roteiro gigante [16], é necessário criar-se, inicialmente, uma seqüência de nós, o roteiro gigante, o que pode ser feito utilizando-se qualquer algoritmo para o problema do caixeiro viajante. O primeiro ponto de demanda desse roteiro é imposto como sendo a base, representada como nó R(0). Seguem-lhe os nós R(1), R(2),..., R(N) no roteiro.

Inicialmente utilizou-se o algoritmo dois ótimo para otimizar um roteiro aleatório fornecido. Posteriormente, optou-se por utilizar um procedimento combinado, no qual um roteiro inicial é criado pelo algoritmo para problemas de caixeiro viajante vizinho mais próximo, sendo o mesmo posteriormente otimizado pelo algoritmo dois ótimo. Com esse procedimento, conseguiu-se uma redução significativa na duração da criação do roteiro gigante e uma diminuição na distância total do mesmo.

Optou-se, para simplificar a implementação computacional, por incluir a base no conjunto de nós a serem roteados para criação do roteiro gigante, diferentemente do algoritmo de partição múltipla de roteiro gigante, pois é indiferente ao algoritmo qual dos nós do roteiro será o imediatamente posterior à base (o primeiro nó a ser visitado na primeira viagem criada), podendo essa definição ser deixada a critério do procedimento de criação do roteiro gigante.

⇒ DETERMINAÇÃO DO NUMERO DE ITERAÇÕES A SEREM REALIZADAS

O procedimento de partição do roteiro gigante é repetido M vezes, deslocando-se o nó seguinte à base no roteiro gigante para a última posição, visando obter-se um maior número de soluções, como foi apresentado na descrição do algoritmo de partição do roteiro gigante no capítulo 2. O número M de iterações é obtido pelas equações 4.7 a 4.11, mostradas a seguir,

$$M = \min \left| M1 \; M2 \right| \qquad [4.7]$$

usando-se o veículo v que

$$\begin{cases} C_v = \max \left| C_1 \; C_2 \; .. \; C_{NTV} \right| \\ Vs_v = \max \left| Vs_i \; \forall \; C_i = C_v \right| \; ; \; 1 \le i \le NTV \end{cases} \qquad [4.8]$$

onde M1 é expresso por

$$\sum_{i=1}^{M1} D_{R(i)} \le C_v < \sum_{i=1}^{M1+1} D_{R(i)} \qquad [4.9]$$

e M2 é obtido por

$$Z_{R(M2)} \le TV < Z_{R(M2+1)} \qquad [4.10]$$

onde

$$Z_m = \frac{d_{0R(1)} + \left(\sum_{i=1}^{R(m-1)} d_{R(i)R(i+1)} \right) + d_{R(m)0}}{Vs_v} + \sum_{i=1}^{m} D_{R(i)} \cdot \frac{Tc_{R(i)} + Tc_0}{Tc_{R(i)} \cdot Tc_0} \qquad [4.11]$$

A metodologia utilizada para cálculo do número de iterações é a mesma proposta por Golden et alii [16], acrescida porém de uma restrição de duração máxima de jornada (equações 4.10 e 4.11), pois a mesma também limita o número máximo de pontos de demanda atendidos pela maior primeira viagem possível do roteiro gigante, junto com a restrição de capacidade.

➲ CRIAÇÃO DE REDE ORIENTADA COM FATOR REDUTOR DE CUSTO FIXO

A partição do roteiro gigante é feita através da criação de uma rede orientada, composta por N+1 nós, cada um dos N primeiros nós correspodendo aos N pontos de demanda do problema, correspodendo o último nó à base.

Sendo dados dois nós R(i) e R(j) da rede orientada, $1 \leq i < j \leq N+1$, onde R(N+1)=base, o custo C_{IJ} do arco unindo esses dois nós é dado por C_{IJ}, representando o custo de um veículo, partindo da base, percorrer o roteiro gigante do i-ésimo ao (j-1)-ésimo nó, utilizando-se o veículo v ($1 \leq v \leq NTV$) para realizar a viagem, conforme as equações 4.12 a 4.14,

$$\text{minimizar } Cij = F.Cf_v + Co_v \cdot \left(d_{0R(i)} + \left(\sum_{m=i}^{j-2} d_{R(m)R(m+1)} \right) + d_{R(j-1)0} \right) \qquad [4.12]$$

sujeito a

$$\sum_{m=i}^{j-1} D_{R(m)} \leq C_v \qquad [4.13]$$

$$\sum_{m=i}^{j-1} D_{R(m)} \cdot \left(\frac{Tc_{R(m)} + Tc_0}{Tc_{R(m)} \cdot Tc_0} \right) + \frac{d_{0R(i)} + \sum_{m=i}^{j-2} d_{R(m)R(m+1)} + d_{R(j-1)0}}{Vs_v} \leq TV \qquad [4.14]$$

sendo C_{IJ} infinito se não houver nenhum tipo de veículo capaz de realizar essa viagem sem violar as restrições 4.13 e 4.14.

As restrições 4.13 e 4.14 são cópias das restrições 4.2 e 4.3, porém o cálculo de C_{IJ} difere da metodologia apresentada anteriormente neste capítulo na inclusão de um fator redutor de custo fixo, F, na equação 4.12.

O conceito do fator redutor de custo fixo se baseia na constatação de que o custo fixo de uma viagem realizada por um veículo alocado exclusivamente à mesma será maior que o custo fixo da mesma viagem à qual for alocado o mesmo veículo, o qual, porém, realiza outras viagens. Por exemplo, supondo que um veículo realize duas viagens, uma das quais ocupa 60% do tempo total de operação do veículo e a outra 40%, o custo fixo da primeira viagem será 60% do custo fixo do veículo no período total de operação, e o da segunda viagem, 40%.

A tendência natural do algoritmo de partição do roteiro gigante é de criar viagens longas e de grande demanda, pois com isso ocorre uma redução do número total de veículos, e, conseqüentemente, no custo fixo total da frota.

Além disso, o algoritmo aloca a uma dada viagem o veículo que a realizará mais econômicamente, considerando os custos fixos e variáveis dos veículos, enquanto que a minimização da duração de uma dada viagem conhecida é função da velocidade do veículo, supondo que a duração de transbordo nos pontos de demanda e na base não dependa do tipo de veículo utilizado.

Entretanto, quanto mais demorada for uma viagem, menos tempo terá o veículo para realizar uma outra viagem, devido à duração máxima

especificada. Se o custo fixo de viagens curtas for reduzido no cálculo dos termos C_{IJ}, o algoritmo terá maior probabilidade de escolhê-las.

Dessa forma, a finalidade do fator redutor de custo fixo é de incentivar o algoritmo a privilegiar viagens de pequena duração, através da redução artificial do seu custo fixo, o que implica na redução de seu custo total. O objetivo é fazer que o algoritmo escolha viagens de menor duração do roteiro gigante, (que possuam maior probabilidade de serem agrupadas com outras viagens para atendimento por um único veículo sem violar a restrição de duração máxima de jornada), as quais, à primeira vista, poderiam não ser as mais econômicas, tanto localmente (o algoritmo pode alocar à viagem um veículo mais rápido, não necessáriamente o que realizaria a viagem com menor custo), quanto globalmente (o algoritmo pode acabar obtendo como solução mais viagens do que sem a utilização do fator redutor de custo fixo, conseqüentemente com mais viagens de/para a base).

O fator redutor de custo fixo é uma parcela positiva e menor ou igual a 1 que multiplica o custo fixo no cálculo de C_{IJ}. Com isso, o fator redutor de custo fixo (antes denominado F) para uma viagem que, partindo da base, percorra o roteiro gigante do i-ésimo até o (j-1)-ésimo nós, retornando a seguir à base, utilizando o veículo v ($1 \leq v \leq NTV$), é F_{IJV}, expresso nas equações 4.15 a 4.17,

$$F_{ijv} = \frac{1}{\text{trunc}\left(\frac{TV}{TA_{ij} + TB_{ijv}}\right)} \qquad [4.15]$$

onde

$$TA_{ij} = \sum_{m=i}^{j-1} D_{R(m)} \cdot \frac{Tc_{R(m)} + Tc_0}{Tc_{R(m)} \cdot Tc_0} \qquad [4.16]$$

$$TB_{ijv} = \frac{d_{0R(i)} + \sum_{m=i}^{j-2} d_{R(m)R(m+1)} + d_{R(j-1)0}}{V\sigma_v} \qquad [4.17]$$

onde trunc[x] é o maior inteiro menor ou igual a x.

Pela equação 4.15, o custo fixo de uma viagem é reduzido à metade se o veículo conseguir realizar outra viagem na duração máxima de operação permitida, a um terço se conseguir realizar mais duas, e assim por diante.

O fator redutor de custo fixo não é contínuo em favor da segurança, para garantir a escolha de uma viagem apenas se a mesma tiver um bom potencial de ser combinada.

➲ PARTIÇÃO DO ROTEIRO GIGANTE

A função dessa etapa é, utilizando-se algum algoritmo para o cálculo de caminho de mínimo custo, como o de Dijkstra, mencionado no trabalho de Bodin et alii [2], determinar qual o caminho de mínimo custo entre o primeiro e o N+1-ésimo nós da rede orientada. Os nós que fizerem parte do caminho de menor custo serão os nós nos quais se iniciará uma nova viagem. Esse procedimento é idêntico ao procedimento de partição do algoritmo de partição do roteiro gigante, apresentado no capítulo 2.

➲ COMBINAÇÃO DE VIAGENS

Com a inclusão do fator redutor de custo fixo podem ser criadas viagens com potencial para serem agrupadas com outras, passando a serem

atendidas por um único veículo. Essas viagens, entretanto, ainda não foram agrupadas, sendo realizadas por veículos distintos.

O procedimento de combinação de viagens, como o próprio nome já diz, procura combinar as viagens obtidas pela partição do roteiro gigante, alocando a cada agrupamento viável o veículo que consiga atender a todas as viagens agrupadas com o menor custo. Cabe observar que, ao término de cada uma das viagens agrupadas, o veículo retornará à base, devendo, portanto, a sua capacidade ser maior ou igual à demanda de qualquer uma das viagens, e a sua velocidade ser suficiente para atender a todas elas na duração máxima especificada para a operação do veículo.

O procedimento realiza uma ordenação das viagens obtidas pela partição do roteiro gigante, criando uma lista, agrupando-se cada viagem da lista com a viagem, dentre as seguintes, que propicie uma maior redução no custo total da frota. Após cada agrupamento realizado, procura-se alocar às viagens combinadas outras viagens, até que nenhum outro agrupamento seja possível. O critério de ordenação adotado será apresentado após a exposição do funcionamento do procedimento de combinação de viagens.

O procedimento é repetido com a próxima viagem da lista que ainda não tenha sido agrupada, e assim por diante, até que se esgotem todos os agrupamentos possíveis. O diagrama de blocos da figura 4.2 ilustra o procedimento de combinação de viagens.

Passo 1: Ordenar as viagens, criando uma lista;

Passo 2: Retirar a primeira viagem da lista, ou a viagem da lista seguinte à atual.

Passo 3: Considerar como custo total do veículo que atenda a essa viagem o custo fixo do mesmo somado ao custo variável de

todas as viagens atendidas pelo mesmo. Considerar como duração total de operação do veículo a somatória da duração de todas as viagens atendidas pelo mesmo;

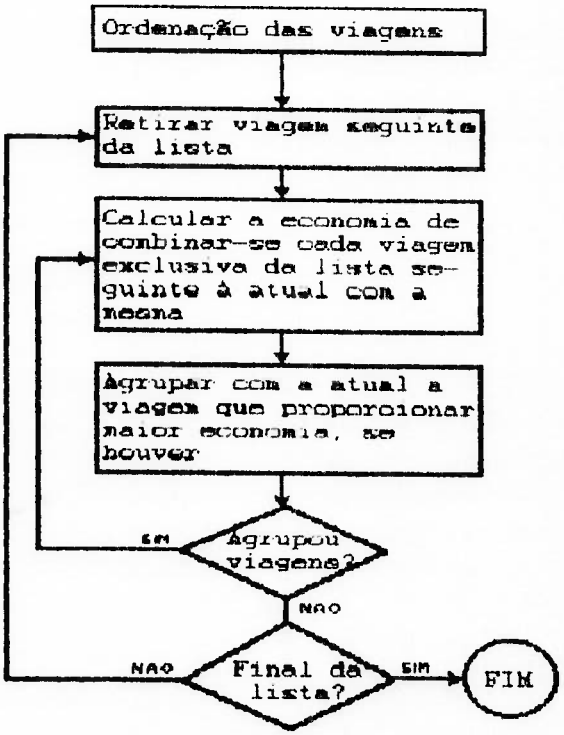


FIGURA 4.2 - PROCEDIMENTO DE COMBINAÇÃO DE VIAGENS

Passo 4: Para todas as viagens da lista seguintes à atual, e que sejam exclusivas, (isto é, que sejam atendidas por um veículo exclusivo, que não atenda a outras viagens), determinar qual tipo de veículo, se houver, atenda a essa viagem, e mais todas as viagens do veículo obtido no passo 3, com o menor custo total, sem violar restrições de capacidade e duração máxima de operação. Calcular a economia obtida com a combinação (custo total do veículo obtido no passo 3, somado ao custo total da viagem exclusiva considerada, subtraído do custo total da combinação);

Passo 5: Agrupar com as viagens do passo 3 a viagem que proporcionou maior economia no passo 4, utilizando-se o veículo que propiciou essa economia, se tiver sido obtida no passo 4 pelo menos uma combinação possível de viagens;

Passo 6: Caso tenha sido possível combinar viagens, retornar ao passo 3;

Passo 7: Caso haja uma viagem seguinte à atual na lista, retornar ao passo 2. Caso contrário, encerrar o procedimento.

O procedimento de ordenação das viagens escolhido foi a ordenação por demandas decrescentes, e, para uma mesma demanda, a ordenação decrescente das velocidades dos veículos que as realizam. Esse procedimento foi adotado pelos seguintes motivos:

- A ordenação segundo demandas decrescentes visa garantir que todas as viagens posteriores a uma qualquer tenham demanda menor ou igual à da viagem em questão;

- A ordenação pela velocidade visa dar preferência das combinações aos veículos mais rápidos, que possuam maiores custos associados, com o objetivo de "diluir" o seu custo fixo, elevado, em mais de uma viagem.

- **PÓS-PROCESSAMENTO DA SOLUÇÃO (PARTIÇÃO DE NOVO ROTEIRO GIGANTE)**

Devido ao fator redutor do custo fixo, algumas viagens podem ter o seu custo fixo artificialmente reduzido se as mesmas tiverem uma duração menor que metade da duração máxima permitida, expressando essa redução o potencial que essas viagens tem para serem agrupadas com outras para serem atendidas por um único veículo. Essa redução do custo fixo visa, no algoritmo de dimensionamento de frota, dar preferência às viagens curtas. Pode ocorrer, entretanto, que uma ou mais dessas viagens não venham a ser agrupadas, o que implica em algumas viagens curtas e de pequena demanda total atendidas por veículos exclusivos, afastando a

solução da ótima. Para minimizar esse problema, "diluindo" os efeitos da utilização do fator redutor de custo fixo, todos os nós pertencentes a viagens atendidas por veículos exclusivos devem ser submetidos a um novo procedimento de criação e partição de roteiro gigante, sem a utilização do fator redutor de custo fixo no cálculo dos termos C_{IJ}.

A solução final apresentada pelo algoritmo, então, apresenta viagens:

- Realizadas por veículos que atendem a outras viagens, obtidas nos passos 2 a 9 do algoritmo;
- Realizadas por veículos exclusivos, obtidas nos passos 10 a 16 do algoritmo.

4.4- COMENTÁRIOS FINAIS

O algoritmo de dimensionamento de frota foi implementado em um programa de computador, comentado e apresentado no apêndice, em linguagem Pascal.

A implementação computacional foi testada para depuração de erros, e depois procedeu-se a testes comparativos e de desempenho, apresentados no capítulo 7.

Entretanto, da maneira como foi formulado, o algoritmo de dimensionamento de frota é incapaz de lidar com uma frota limitada em número de veículos disponíveis de cada tipo (frota fornecida), tendo sido, para resolver esse problema específico, desenvolvido o algoritmo de alocação de frota, apresentado no capítulo 5.

5- ALGORITMO PARA O PROBLEMA DE ALOCAÇÃO DE FROTA

5.1- FINALIDADE DO ALGORITMO

No capítulo anterior foi apresentado o algoritmo de dimensionamento de frota, que tem por objetivo selecionar, a partir de uma série de tipos de veículos disponíveis, uma frota de veículos para atender a um conjunto de pontos de demanda com o mínimo custo total (fixo + variável).

Esse algoritmo pressupõe que são disponíveis tantos veículos de cada tipo quantos forem necessários, sendo que tal pressuposto não é válido no caso de ser, por um ou outro motivo, possível adquirir quantidades limitadas de cada tipo disponível, ou de dispor-se de uma frota conhecida, que deve ser utilizada.

Para resolver problemas nos quais é fornecida uma frota de veículos, e não uma lista de tipos de veículos que podem ser adquiridos, foi criado o algoritmo de alocação de frota.

5.2- FORMULAÇÃO DO ALGORITMO

Dada uma frota de veículos, o algoritmo de alocação de frota utilizará a mesma para atender a um conjunto de pontos de demanda, conceito diferente do do algoritmo de dimensionamento de frota, que determina a composição da frota para atender aos pontos de demanda referidos.

O objetivo do algoritmo é alocar todos os pontos de demanda à frota fornecida com o menor custo variável possível, partindo do pressuposto que já existe uma frota de veículos, e, dessa forma, o custo fixo da mesma é pago independentemente da sua alocação a uma dada missão.

Procedimentos heurísticos nem sempre encontram a melhor solução. Se para o algoritmo de dimensionamento de frota isso implica na possibilidade de uma solução não ótima, para o algoritmo de alocação de veículos, além de não se garantir a alocação ótima, também pode ocorrer a existência de pontos de demanda órfãos ao final do problema, não atendidos por nenhum veículo, devido à alocação imperfeita dos outros pontos. Para tanto foi definido o conceito de deficit da frota, um valor que expressa, em um dado instante do processamento do algoritmo, qual o excesso de demanda ainda não alocada em relação à capacidade estática dos veículos ainda não utilizados. Um déficit negativo implica em maior capacidade estática de transporte disponível que demanda não atendida, sendo objetivo do algoritmo minimizar o déficit do problema, se possível tornando-o nulo ou negativo.

O algoritmo desenvolvido seleciona um dentre diversos veículos fornecidos, alocando pontos de demanda ao mesmo, prosseguindo até que nenhuma alocação adicional seja possível, partindo, então, para a alocação em outro veículo, se houver, e se ainda existirem pontos de demanda não alocados a nenhum veículo. Esse procedimento é extremamente flexível, permitindo fácilmente a inclusão de restrições, e de execução computacional muito rápida. Por outro lado, essa sistemática é míope, concentrando-se em um único veículo, ignorando os efeitos que uma alocação possa ter nas alocações futuras de outros poontos de demanda a esse ou a outros veículos.

Para reduzir os efeitos dessa miopia, foi implementado um procedimento de melhoria derivado do algoritmo de varredura de Gillett &

Miller [13], que consiste em substituir-se nós alocados em viagens por nós ainda não alocados, sempre que isso satisfizer o objetivo de diminuir o déficit.

Tratando-se de um algoritmo heurístico, é possível que ao final do processo existam um ou mais pontos de demanda que não foram alocados a nenhum veículo (alocação encerrada com déficit > 0), particularmente em problemas com poucas soluções viáveis. O algoritmo encerra, nesse caso, apontando quais pontos de demanda deixaram de ser atendidos. Na implementação computacional do algoritmo de alocação de frota, entretanto, se ao final da execução existirem pontos de demanda órfãos, o programa executa o programa de dimensionamento de frota para os mesmos, obtendo um ou mais veículos adicionais à frota existente, tendo isso sido feito para permitir a comparação das soluções obtidas pelos algoritmos de dimensionamento de frota e de alocação de frota.

O algoritmo resultante será exposto e detalhado ao longo desse capítulo.

5.3- ALGORITMO PARA O PROBLEMA DE ALOCAÇÃO DE FROTA

O diagrama de blocos do algoritmo de alocação de frota é apresentado nas figuras 5.1 a 5.3.

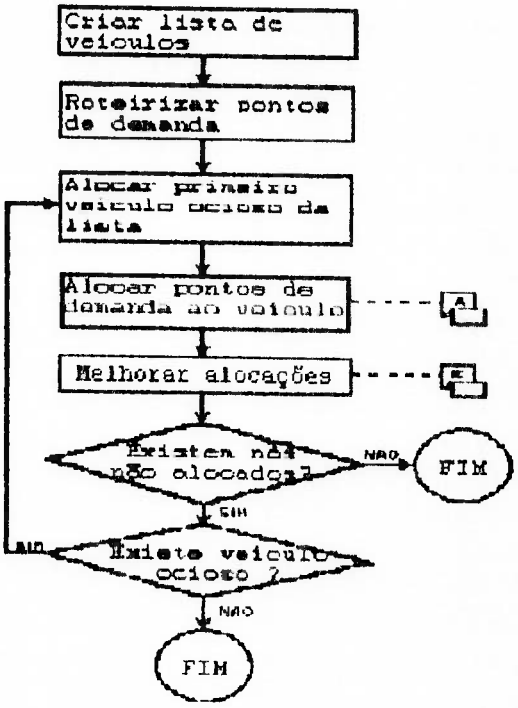


FIGURA 5.1
ROTINA PRINCIPAL

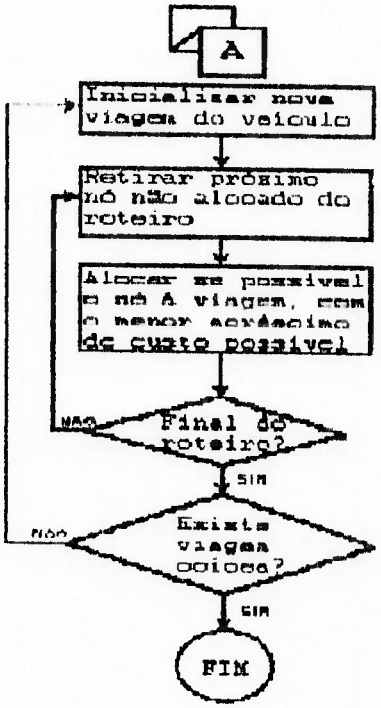


FIGURA 5.2
ALOCAÇÃO DOS PONTOS DE DEMANDA

➲ Inicialização do algoritmo: Consiste em:

- Entrada dos dados da frota de veículos; Descrição da capacidade, velocidade e custo variável de cada veículo da frota; Cálculo do custo unitário de cada veículo, obtido pela divisão do custo variável do veículo pela sua capacidade (esse índice permite a seleção dos veículos mais econômicos); Ordenação dos veículos segundo custos unitários crescentes, e, para um mesmo custo unitário, capacidades de carga decrescentes (inicialmente serão alocados os veículos mais econômicos e os maiores);
- Entrada dos dados dos pontos de demanda e da base; Decomposição de nós com excesso de demanda em nós fictícios, sendo utilizado o veículo de menor capacidade disponível, ao invés do de menor custo (como adotado no mesmo procedimento para o algoritmo de dimensionamento de frota) para a divisão, para que os nós fictícios

possam ser atendidos por qualquer veículo disponível; Descrição da localização e da taxa de embarque de todos os pontos de demanda e da base; Descrição da demanda de cada ponto; Descrição da duração máxima admissível para a operação dos veículos; Roteirização dos pontos de demanda;

- Alocação de pontos de demanda a veículos: Dado um veículo, alocar nós ao mesmo, criando para tanto quantas viagens forem necessárias, com a soma da duração das mesmas não ultrapassando a duração máxima de operação admissível para os veículos. Retirar o primeiro nó não alocado do roteiro gigante. Para cada ponto de demanda, percorrer todas as viagens existentes do veículo, verificando o custo e a possibilidade de inserir-se o mesmo entre cada dois nós do roteiro do veículo (pontos de demanda ou base), obedecendo às restrições de duração máxima de operação e de capacidade máxima do veículo. Alocar o ponto de demanda entre dois nós tais que o custo de inserção seja o menor. Se não puder alocar o nó em nenhuma das viagens existentes do veículo, criar uma viagem nova, alocando o ponto de demanda. Repetir o procedimento para todos os nós não alocados, enquanto for possível a alocação. O diagrama de blocos da Figura 5.2 mostra o funcionamento desse procedimento.

- Melhoria das alocações: Esse procedimento tem duas fases, que são repetidas até que nenhuma melhoria adicional seja possível nas viagens existentes. Na primeira fase, seleciona-se o ponto de maior demanda não alocado e procura-se incluí-lo no roteiro de qualquer veículo que tenha nós alocados, através do desalocamento do nó de menor capacidade possível em lugar do qual possa ser incluído o ponto selecionado, sem violar as restrições de capacidade do veículo e de duração máxima de operação. Na segunda fase aplica-se o procedimento de alocação de pontos de demanda, descrito anteriormente, em todos os veículos que possuam pelo menos um nó alocado, visando a alocação de mais pontos de demanda. O procedimento de melhoria de alocações é representado no diagrama de blocos da Figura 5.3.

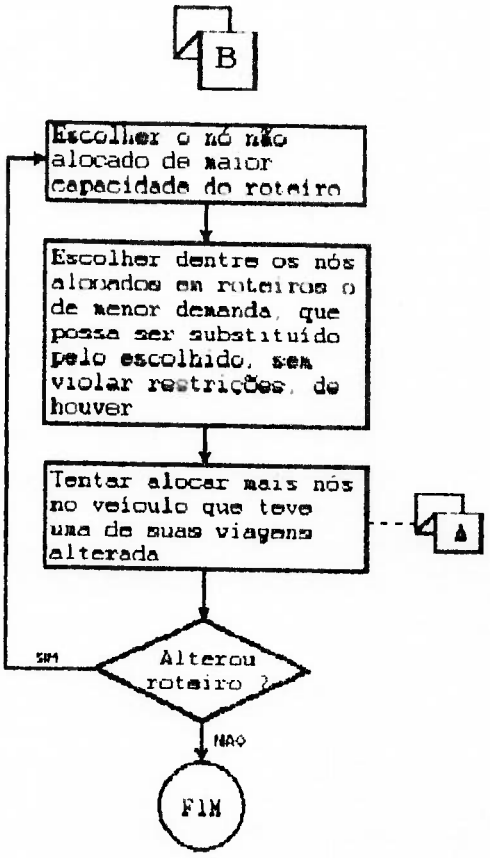


FIGURA 5.3
MELHORIA DAS ALOCAÇÕES

Com esse procedimento procura-se forçar a alocação de nós de grande demanda, que tem maior dificuldade de alocação em viagens que pontos de menor demanda. Além disso, simultaneamente, procura-se diminuir a ociosidade de capacidade de outras viagens, com a alocação de nós de pequena demanda, desalocados de outras viagens, com maior potencial de alocação em outras viagens que nós de maior demanda. Em suma, esse procedimento otimiza as viagens existentes através do maior aproveitamento da capacidade dos veículos, diminuindo a capacidade ociosa de cada viagem.

5.4- DESENVOLVIMENTO DO ALGORITMO

Nessa seção será comentado o desenvolvimento de quatro procedimentos do algoritmo de alocação de frota, que foram objeto de constantes alterações devido à análise de resultados computacionais.

O algoritmo proposto no item anterior é a versão final do mesmo, utilizada para os testes computacionais levados a cabo no capítulo 7.

Em se tratando de um algoritmo novo e iterativo, a versão final revelou-se bem diferente da primeira versão implementada em computador, tendo sido efetuadas diversas alterações por força de análise de resultados computacionais obtidos na resolução de problemas.

Logo, a implementação computacional foi fundamental para o aprimoramento do algoritmo, tendo sido testadas diversas alterações, muitas das quais foram incorporadas ao algoritmo, por melhorarem sensivelmente o seu desempenho.

➲ CRITÉRIO DE ORDENAÇÃO DE PONTOS DE DEMANDA

Nenhum dos critérios adotados para a ordenação dos pontos de demanda, dentre os tentados (segundo capacidade, segundo distância em relaçao à base, através da construção de um roteiro gigante, e de maneira aleatória), apresentou-se superior aos demais em termos de qualidade de solução. Escolheu-se a ordenação através da construção de um roteiro gigante para uma maior compatibilidade entre as soluções obtidas pelo algoritmo de dimensionamento de frota (baseado em um roteiro gigante) e pelo algoritmo de alocação de frota.

➲ CRITÉRIO DE ORDENAÇÃO DOS VEÍCULOS DA FROTA

O algoritmo mostrou-se mais sensível a alterações no procedimento de ordenação de veículos que no procedimento de ordenação de pontos de demanda, pois os pontos de demanda alocados a um veículo são

primeiramente função do veículo retirado da lista. Os seguintes critérios foram considerados:

- ordenação segundo menor velocidade dos veículos, com maior capacidade como critério secundário (visando-se a redução do custo variável da frota, com a utilização primeiro de veículos mais lentos, de custo variável menor);
- ordenação segundo menor custo unitário (custo variável dividido pela capacidade), com maior capacidade como critério secundário (também visando a redução do custo variável da frota);
- ordenação segundo menor capacidade dos veículos, com menor velocidade como critério secundário (visando-se a alocação inicialmente de veículos que apresentem menor potencial de alocação de pontos de demanda, numa tentativa para minimizar o número de pontos de demanda não atendidos);
- ordenação segundo maior velocidade dos veículos, com maior e menor capacidade como critério secundário (visando-se utilizar veículos mais rápidos na tentativa de reduzir o número de veículos necessários para atender o problema, objetivando reduzir o custo total);
- ordenação segundo maior capacidade dos veículos, com maior velocidade como critério secundário (o oposto da anterior : alocação inicial em veículos que possam atender mais pontos de demanda, para minimizar o número de pontos de demanda não atendidos);
- ordenação aleatória;

A tabela 5.1 foi obtida resolvendo-se 100 problemas aleatórios (com características fornecidas nas tabelas 5.2 e 5.3) para cada critério de ordenação, com frota gerada pelo algoritmo de dimensionamento de frota (frota A), ou gerada pelo próprio algoritmo de alocação de frota,

tornando-se infinito o numero de veículos de cada tipo disponíveis para a composição da frota (frota B). ↑ significa crescente e ↓ decrescente.

TABELA 5.1 - COMPARAÇÃO DE CRITÉRIOS DE ORDENAÇÃO

	Critérios de ordenação		Frota A		Frota B	
	Primário	Secundário	Custo	N°veic	Custo	N°veic
1	velocidade ↓	capacidade ↑	15979	6.86	26290	6.59
2	custo ↓	capacidade ↑	16062	6.77	18334	8.50
3	capacidade ↓	velocidade ↓	16004	6.77	17977	9.08
4	velocidade ↑	capacidade ↑	15947	6.85	31634	5.20
5	velocidade ↑	capacidade ↓	16017	6.79	21421	7.11
6	capacidade ↑	velocidade ↑	15965	6.89	31634	5.20
7	aleatório	----	16084	6.81	21364	6.62

TABELA 5.2 - TIPOS DE VEÍCULOS DISPONÍVEIS

Capacidade (unidades)	Velocidade (km/h)	Custo	
		Fixo (Us$)	Variável (Us$/km)
200	33	$769.49	$1.56
200	41	$917.71	$1.73
200	46	$1,108.29	$2.12
250	33	$964.38	$1.95
250	41	$1,156.25	$2.22
250	46	$1,333.16	$2.54
300	33	$1,184.87	$2.44
300	41	$1,421.07	$2.79
300	46	$1,597.00	$3.07
350	33	$1,398.45	$2.94
350	41	$1,701.64	$3.43
350	46	$1,908.71	$3.77
400	33	$1,608.66	$3.45
400	41	$1,976.89	$4.08
400	46	$2,224.26	$4.48
450	33	$1,819.63	$3.96
450	41	$2,247.15	$4.75
450	46	$2.533.20	$5.21
500	33	$2,031.31	$4.48
500	41	$2,518.05	$5.43
500	46	$2,839.30	$5.96

TABELA 5.3 - CARACTERÍSTICAS DO PROBLEMA

- Número de nós	20	
- Raio médio dos pontos de demanda (R)	50,60,...,90	km
- Média da demanda	60,125,...,320	kg
- Desvio padrão da demanda	10,20,...,50	kg
- Média da taxa de embarque	250	kg/h
- Desvio padrão da taxa de embarque	5	kg/h
- Posição relativa da base	$0, \frac{\sqrt{2}}{2}R, \ldots, 2\sqrt{2}\ R$	km
- Duração máxima de jornada	20,21,...,24	h

Observa-se que, quando a frota é fornecida pelo programa de dimensionamento de frota, todos os critérios apresentam soluções próximas, inclusive a ordenação aleatória, com menos de 0.7% de diferença no custo entre o melhor e o pior critérios. Também o número de veículos da frota manteve-se estável, a alteração do critério de ordenação de veículos não causando grandes alterações no número de problemas em que a frota gerada pelo algoritmo de dimensionamento de frota revelou-se insuficiente para o algoritmo de alocação de frota atender os pontos de demanda.

Por outro lado, quando o algoritmo de alocação de frota teve a incumbência de dimensionar a frota (frota B), optando entre um número ilimitado de veículos de cada um dos tipos disponíveis de veículos, o critério de ordenação passou a ser extremamente importante, ocorrendo uma variação de custo de quase 76% entre o pior e o melhor critério de ordenação. Conclue-se, então, que o critério de ordenação ajuda a minimizar o custo total do problema a ser resolvido, quando a frota fornecida é folgada (mais que suficiente para atender a todos os pontos

de demanda do problema) ou quando o algoritmo de alocação de frota é usado para o dimensionamento de frota.

Tanto o critério de ordenação segundo menor custo unitário quanto o critério de ordenação segundo menor capacidade tiveram bom comportamento, com frota restrita (frota A) e dimensionando uma frota (frota B). Na dissertação adotou-se a ordenação segundo menor custo unitário, devido a resultados melhores obtidos por esse critério na fase de desenvolvimento do algoritmo de alocação de frota

⇒ PROCEDIMENTO DE ALOCAÇÃO DE PONTOS DE DEMANDA AOS VEÍCULOS DA FROTA

Inicialmente a alocação de pontos de demanda a veículos era feita levando-se em conta a redução no déficit de capacidade que a mesma trazia. A intenção era minimizar a ocorrência de pontos de demanda que não fossem atendidos ao final do processamento. Entretanto, esse procedimento acabava levando a roteiros pouco eficientes, nos quais muito tempo de viagem era perdido pelos veículos no deslocamento entre os nós. Dessa forma, se por um lado as viagens tendiam a ter uma maior utilização da capacidade dos veículos, por outro lado menos viagens eram feitas pelos veículos devido à restrição de duração máxima de operação dos veículos, o que no final acabava resultando em um grande potencial de nós não alocados ao final do processamento, e com grande tempo de processamento gasto na otimização dos roteiros dos veículos. Dessa forma, optou-se por alocar os nós a veículos objetivando o menor custo de inserção possível, deixando-se para o procedimento otimizador de soluções a tarefa de minimizar o déficit de capacidade da frota, conseguindo-se com isso simultâneamente uma redução no tempo de processamento, no custo da solução e na probabilidade de nós não alocados ao fim do processamento.

⇒ PROCEDIMENTO DE MELHORIA DE ALOCAÇÕES

O procedimento de melhoria de alocações era inicialmente invocado uma única vez ao término do processo de alocação de pontos de demanda aos veículos, que tinha por objetivo melhorar os roteiros dos veículos, visando a redução do custo da solução e a inclusão de pontos de demanda porventura não atendidos nos roteiros, sempre que o déficit fosse reduzido ou mantido constante com o processo. Era um procedimento derivado do algoritmo dois-ótimo de melhoria de roteiros, bastante demorado (pela própria concepção iterativa do dois-ótimo e pela baixa qualidade dos roteiros obtidos pelo processo de alocação de pontos de demanda), e que não apresentava significativa redução no número de pontos de demanda não atendidos ao final do processo, apesar de conseguir redução no custo total. A primeira alteração foi a execução do procedimento após cada inclusão de pontos de demanda em um novo veículo, e não mais uma única vez ao final da alocação de todos os veículos. Entretanto, apesar do substancial aumento do tempo de processamento do algoritmo (que chegava a ser quatro vezes mais lento que o algoritmo de dimensionamento de frota, dependendo dos critérios de ordenação adotados), não conseguiu-se melhora significativa no custo da solução nem diminuição no número de pontos de demanda não alocados em relaçao aos que eram obtidos com uma única passagem do algoritmo de otimização. Optou-se, portanto, por alocar pontos de demanda aos veículos visando-se roteiros eficientes, e minimizando-se o déficit no processo de melhoria, que foi inteiramente refeito. Abandonou-se o algoritmo dois-ótimo como base, partindo-se para a utilização de um conceito apresentado no algoritmo de varredura de Gillett & alii [13], a substituição de nós de um roteiro por nós não alocados em nenhum roteiro, sempre quando isso satisfizer a um objetivo (no caso, redução do déficit), sem violar quaisquer restrições. Passou-se a melhorar, através desse procedimento, todos os roteiros de todos os veículos ao término do processo de alocação de pontos de demanda a um veículo, e obteve-se sensíveis reduções na probabilidade de um ou mais pontos de demanda não serem

alocados ao final do processamento, (sendo que na maioria dos casos em que isso ocorreu foi um único nó não alocado, e de pequena demanda, conforme apresentado pelos resultados computacionais), sem ocorrer aumento no custo total quando comparado aos procedimentos anteriores, e com uma grande diminuição no tempo de processamento, passando o algoritmo de alocação de frota a ter duração de execução comparável com o de dimensionamento de frota, mesmo com a ordenação dos pontos de demanda através da construção de um roteiro gigante (utilizando-se o procedimento dois-ótimo para tanto).

5.5- COMENTÁRIOS FINAIS

O algoritmo apresentado neste capítulo revelou-se de implementação computacional fácil e de execução rápida na resolução de problemas para os quais foi formulado (restritos e com frota fornecida).

Entretanto, o problema dos pontos de demanda órfãos não pôde ser eliminado, particularmente em problemas com poucas soluções possíveis viáveis. A implementação computacional, descrita no apêndice, permite, além de apontar os pontos de demanda não alocados a nenhum veículo, que novos veículos sejam adicionados à frota, através da execução do programa de dimensionamento de frota. Entretanto, esse procedimento incorporará novos veículos à frota, quando muitas vezes isso pode não ser necessário. Uma nova heurística, o algoritmo de melhoria de soluções, foi desenvolvida para, partindo de uma solução fornecida, através de um procedimento combinatório, minimizar o seu custo variável, podendo também minimizar a ocorrência de pontos de demanda órfãos. Esse algoritmo é apresentado no capítulo sexto.

6- ALGORITMO DE MELHORIA DE SOLUÇÕES

6.1- INTRODUÇÃO

Devido à diferença de concepção dos algoritmos de dimensionamento e de alocação de frota, constatou-se experimentalmente que em alguns casos a frota obtida pelo primeiro era insuficiente para o segundo obter uma solução viável no mesmo cenário, gerando pontos de demanda órfãos, e que isso devia-se não a erros na implementação do algoritmo de alocação de frota, mas pela incapacidade do mesmo de obter uma solução viável em problemas com poucas dentre as soluções possíveis viáveis.

Também a solução obtida pelos algoritmos de dimensionamento e de alocação de frota, em se tratando de heurísticas, pode estar distante da solução de menor custo.

O algoritmo de melhoria de soluções foi desenvolvido para minimizar o custo total de uma solução inicial fornecida, objetivando uma redução do custo variável, tendo, entretanto, potencial para reduzir o custo fixo da frota através da eliminação de veículos ociosos, isto é, que não atendam a nenhum ponto de demanda.

Partindo de uma solução inicial viável, o algoritmo, através de um procedimento combinatório, procura obter soluções que se aproximem mais da ótima do que a original, cessando o processamento quando nenhuma melhoria adicional puder ser conseguida pela metodologia utilizada.

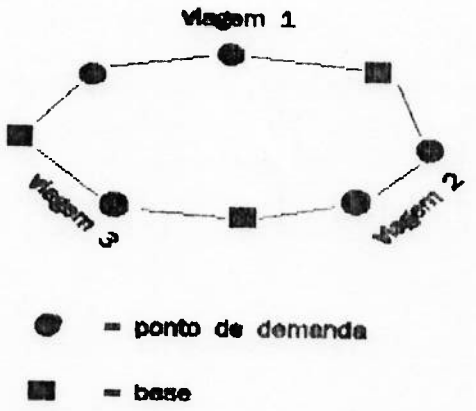
Dessa forma, o algoritmo:

- Necessita de uma solução inicial viável;
- Obtém uma solução de custo total igual ou menor que a inicial;

6.2- FORMULAÇÃO DO ALGORITMO DE MELHORIA DE SOLUÇÕES

O algoritmo desenvolvido é um procedimento combinatório derivado do algoritmo 2-ótimo para melhoria de roteiros, que tem por objetivo a diminuição do custo variável da solução inicial fornecida.

FIGURA 6.1 - ROTEIRO GIGANTE COM NÓS AGRUPADOS EM VIAGENS



No algoritmo 2-ótimo, um roteiro gigante é composto de uma série de nós, cada nó tendo um nó anteiror e um posterior no roteiro. No algoritmo de melhoria de soluções, o roteiro gigante é composto por uma série de nós agrupados em viagens conectadas, sendo o primeiro e o último nós de cada viagem, necessáriamente, a base, sendo os demais os pontos de demanda alocados à essa viagem, como na figura 6.1.

Nenhuma viagem pode violar as restrições do problema. No algoritmo de melhoria de soluções, como no 2-ótimo, os nós são ligados por arcos. Ambos os algoritmos realizam a troca de dois arcos não adjacentes, visando uma melhoria com a mesma (por exemplo, redução de custo), mas enquanto que no algoritmo 2-ótimo a troca é efetivada se houver a melhoria, para que ocorra a efetivação no de melhoria de soluções, a

troca também não pode resultar em uma viagem que viole as restrições do problema.

O algoritmo de melhoria de soluções visa minimizar o custo variável da frota (sua função objetivo). Tem, entretanto, potencial para reduzir o custo fixo da frota, pois, minimizando o custo variável (função da distância percorrida), pode permitir que os veículos, com menor tempo gasto em trânsito, realizem mais viagens, o que pode acabar gerando veículos ociosos. No algoritmo, veículos ociosos são aqueles que realizem viagens compostas apenas por bases, sem nenhum ponto de demanda alocado a elas (não saem da base). Se o algoritmo detecta veículos ociosos, eles são retirados da frota, o que resulta em diminuição do custo fixo.

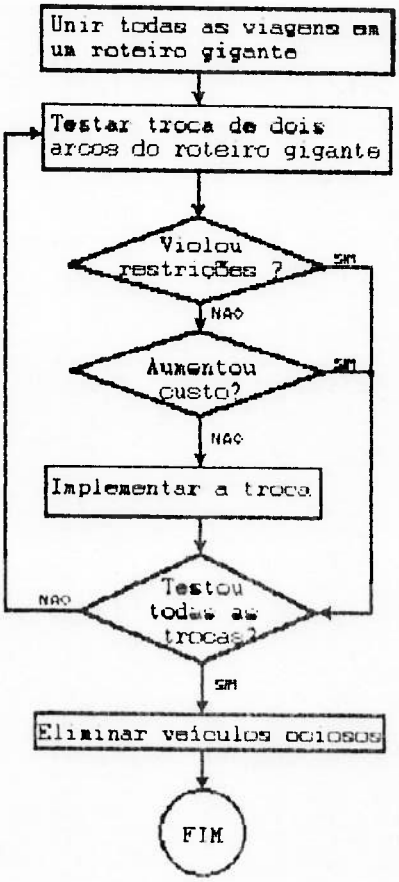
Dessa forma, o algoritmo de melhoria de soluções:

- Tem por função primeira a redução do custo total da solução inicial através da minimização de seu custo variável;
- Pode reduzir o custo fixo da frota, como resultado do esforço de minimização do custo variável, eliminando veículos da composição da frota.

6.3- ALGORITMO DE MELHORIA DE SOLUÇÕES

O algoritmo de melhoria de soluções é esquematizado pelo diagrama de blocos da figura 6.2. Dado um conjunto de viagens, cada viagem realizada por um veículo (que pode ou não ser exclusivo), que inicia e termina a mesma na base, percorrendo um conjunto de pontos de demanda, o algoritmo conecta todas as viagens unindo as bases final de uma viagem e inicial de outra. O resultado é um roteiro gigante composto por pontos de

demanda e bases, cada base representando o início de uma nova viagem. Esse procedimento é representado na figura 6.3.



A melhoria é procurada através da troca simultânea de dois arcos quaisquer da rede, como na figura 6.4. Se, com a troca, nenhuma viagem e nenhum veículo violarem as restrições formuladas, e for conseguida uma redução no custo variável, a troca, e a conseqüênte alteração nos roteiros de antes da troca, será efetivada. Encerra-se o procedimento quando nenhuma melhoria no custo variável puder ser conseguida com a troca simultânea de dois arcos quaisquer da rede.

Por último, localiza-se todos os veículos, se houverem, que, ao final do processo, não atendam a nenhum ponto de demanda, e elimina-se os mesmos da frota, diminuíndo-se o custo fixo.

6.4- COMENTÁRIOS FINAIS

Como mencionado anteriormente, pode ocorrer que o algoritmo de alocação de frota gere pontos de demanda órfãos, usando a mesma frota obtida pelo algoritmo de dimensionamento de frota no mesmo cenário. Isso

pode ser em parte contornado pelo uso do algoritmo de melhoria de soluções. Deve-se alocar aos pontos de demanda órfãos veículos fictícios de custo variável muito alto, se comparado com os demais veículos da frota, e capacidade igual à demanda do nó órfão a ele alocado, e aplicar o algoritmo de melhoria de soluções. Se for conseguida a ocorrência de veículos ociosos, esses serão primeiramente os veículos fictícios, devido ao seu alto custo variável. Se todos os veículos fictícios ficarem ociosos, terá se conseguido atender o problema com a frota obtida pelo algoritmo de dimensionamento de frota. A implementação computacional do algoritmo de melhoria de soluções é apresentada no apêndice.

FIGURA 6.2
ALGORITMO DE MELHORIA DE SOLUÇÕES

FIGURA 6.3 - CONSTRUÇÃO DE ROTEIRO GIGANTE COM VIAGENS

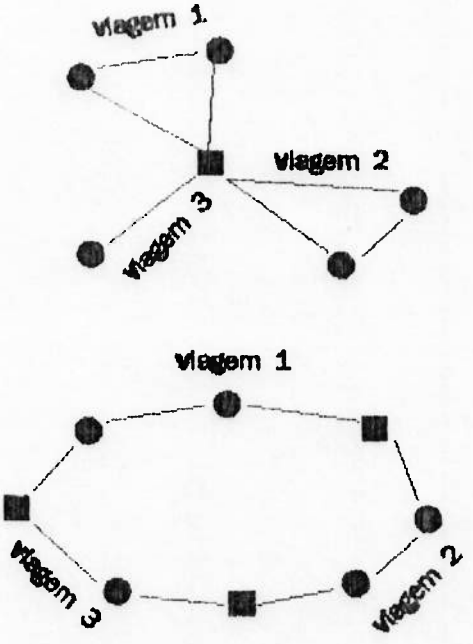
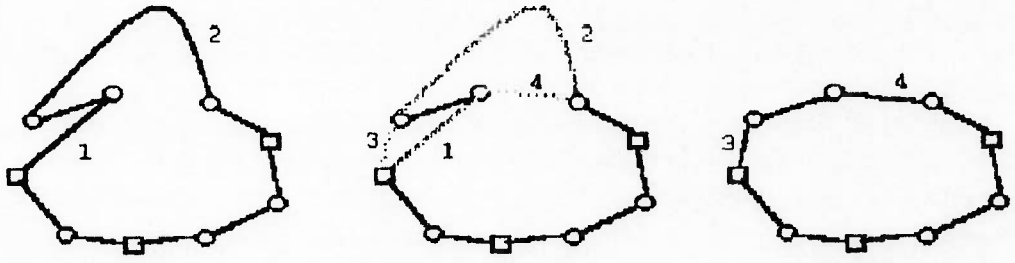


FIGURA 6.4 - PROCESSO DE MELHORIA DO ROTEIRO GIGANTE PELA TROCA DE DOIS ARCOS NÃO ADJACENTES



Roteiro original

Substituição dos arcos 1 e 2 pelos arcos 3 e 4

Roteiro melhorado

7- TESTES COMPUTACIONAIS DOS ALGORITMOS

7.1- INTRODUÇÃO

Nesse capítulo serão apresentados os resultados dos testes comparativos e de desempenho a que foram submetidos os algoritmos desenvolvidos na presente dissertação.

Podemos dividir os testes efetuados em duas séries:

- Resolução de problemas obtidos da literatura estudada e comparação do resultado e do desempenho com os obtidos por outros algoritmos;

- Resolução de problemas baseados no cenário-padrão proposto no capítulo 3, para os quais os algoritmos foram especificamente desenvolvidos;

A primeira série de testes, comparativos com outros algoritmos correntes da literatura, tornou patente a desvantagem dos algoritmos desenvolvidos na resolução de problemas mais gerais em termos de qualidade da solução obtida.

A segunda série mostrou o bom desempenho dos algoritmos na resolução de problemas mais restritos. Observou-se também a não aderência do comportamento dos programas de dimensionamento e de alocação de frota, fato esperado devido à diferença de concepção de ambos.

Os algoritmos foram desenvolvidos em um micro IBM-XT de 8 MHz. Os testes finais foram realizados em um micro IBM-AT de 16 MHz.

7.2- PRIMEIRA SÉRIE DE TESTES: CENÁRIO POUCO RESTRITO

Na primeira série de testes foram utilizados problemas propostos por Christofides & Eilon [5], e utilizados por Fisher & Jaikumar [11] para testes comparativos do desempenho do algoritmo desenvolvido naquele trabalho com outros algoritmos.

Os algoritmos de dimensionamento e de alocação de frota foram submetidos a testes comparativos com os seguintes algoritmos:

- Clarke & Wright (C&W), Clarke & Wright [9];
- Varredura (Sweep), Gillett & Miller [13];
- Fisher & Jaikumar (F&J), Fisher & Jaikumar [11];
- Partiçao de roteiro gigante (PRG), Golden & alii [16].

Os problemas utilizados são baseados em três problemas sugeridos por Christofides & Eilon [5], reproduzidos nas tabelas 7.1 a 7.3, e que foram utilizados nos testes comparativos realizados por Fisher & Jaikumar [11].

TABELA 7.1 - CENÁRIO DE 50 PONTOS

Base : Coordenada X = 30
Coordenada Y = 40

X	Y	Dem cwt	X	Y	Dem cwt	X	Y	Dem cwt	X	Y	Dem cwt
37	52	7	12	42	21	17	33	41	62	63	17
49	49	30	36	16	10	13	13	9	63	69	6
52	64	16	52	41	15	57	51	28	32	22	9
20	26	9	27	23	3	62	42	8	45	35	15
40	30	21	37	69	11	42	57	8	59	15	14
21	47	15	38	46	12	16	57	16	5	6	7
17	63	19	46	10	23	8	52	10	10	17	27
31	62	23	61	33	26	7	38	28	21	10	13
52	33	11	25	32	25	27	68	7	5	64	11
51	21	5	25	55	17	30	48	15	30	15	16
42	41	19	48	28	18	43	67	14	39	10	10
31	32	29	56	37	10	58	48	6			
5	25	23	32	39	5	58	27	19			

TABELA 7.2 - CENÁRIO DE 75 PONTOS

Base : Coordenada X = 40
Coordenada Y = 40

X	Y	Dem cwt	X	Y	Dem cwt	X	Y	Dem cwt	X	Y	Dem cwt
22	22	18	66	14	22	41	46	18	29	39	12
36	26	26	44	13	28	55	34	17	54	38	19
21	45	11	26	13	12	35	16	29	55	57	22
45	35	30	11	28	6	52	26	13	67	41	16
55	20	21	7	43	27	43	26	22	10	70	7
33	34	19	17	64	14	31	76	25	6	25	26
50	50	15	50	30	21	22	53	28	65	27	14
55	45	16	51	42	27	26	29	27	40	60	21
26	59	29	50	15	19	50	40	19	70	64	24
40	66	26	48	21	20	55	50	10	64	4	13
55	65	37	12	38	5	54	10	12	36	6	15
35	51	16	15	56	22	60	15	14	30	20	18
62	35	12	66	8	11	47	66	24	20	30	11
62	57	31	59	5	3	30	60	16	15	5	28
62	24	8	35	60	1	30	50	33	50	70	9
21	36	19	27	24	6	12	17	15	57	72	37
33	44	20	40	20	10	15	14	11	45	42	30
9	56	13	40	37	20	16	19	18	38	33	10
62	48	15	50	4	8	21	48	17			

TABELA 7.3 - CENÁRIO DE 100 PONTOS

Base : Coordenada X = 35
Coordenada Y = 35

X	Y	Dem cwt	X	Y	Dem cwt	X	Y	Dem cwt	X	Y	Dem cwt
41	49	10	45	30	17	49	58	10	49	42	13
35	17	7	35	40	16	27	43	9	53	43	14
55	45	13	41	37	16	37	31	14	61	52	3
55	20	19	64	42	9	57	29	18	57	48	23
15	30	26	40	60	21	63	23	2	56	37	6
25	30	3	31	52	27	53	12	6	55	54	26
20	50	5	35	69	23	32	12	7	15	47	16
10	43	9	53	52	11	36	26	18	14	37	11
55	60	16	65	55	14	21	24	28	11	31	7
30	60	16	63	65	8	17	34	3	16	22	41
20	65	12	2	60	5	12	24	13	4	18	35
50	35	19	20	20	8	24	58	19	28	18	26
30	25	23	5	5	16	27	69	10	26	52	9
15	10	20	60	12	31	15	77	9	26	35	15
30	5	8	40	25	9	62	77	20	31	67	3
10	20	19	42	7	5	49	73	25	15	19	1
5	30	2	24	12	5	67	5	25	22	22	2
20	40	12	23	3	7	56	39	36	18	24	22
15	60	17	11	14	18	37	47	6	26	27	27
45	65	9	6	38	16	37	56	5	25	24	20
45	20	11	2	48	1	57	68	15	22	27	11
45	10	18	8	56	27	47	16	25	25	21	12
55	5	29	13	52	36	44	17	9	19	21	10
65	35	3	6	68	30	46	13	8	20	26	9
65	20	6	47	47	13	49	11	18	18	18	17

Foram, então, gerados 10 problemas diferentes, da seguinte maneira:

- Problema 1: 50 pontos de demanda, sem restrição de duração das viagens, apresentado na tabela 7.1, com veículos de capacidade 8 toneladas;
- Problema 2: 75 pontos de demanda, sem restrição de duração das viagens, apresentado na tabela 7.2, com veículos de capacidade 7 toneladas;

- Problema 3: 100 pontos de demanda, sem restrição de duração das viagens, apresentado na tabela 7.3, com veículos de capacidade 10 toneladas;
- Problema 4: 150 pontos de demanda, sem restrição de duração de viagens, obtidos pela união dos pontos de demanda dos problemas 1 e 3, com base e capacidade do veículo do problema 3;
- Problema 5: 199 pontos de demanda, sem restrição de duração de viagens, obtidos pela união dos pontos de demanda do problema 4 com os 49 primeiros pontos de demanda do problema 2, com base e capacidade do veículo do problema 3.

Para os problemas 6 a 10 foi assumida uma velocidade para os veículos de 1 unidade de distância por unidade de tempo. Além disso 10 unidades de tempo são assumidas como duração da descarga em cada ponto de demanda. Como a demanda de cada ponto difere, foi assumida uma razão de carregamento tal que a demora em cada ponto de demanda seja sempre de 10 unidades de tempo, conforme o trabalho de Fisher & Jaikumar [11].

- Problema 6: Igual ao problema 1, com duração das viagens restrita a 200 unidades de tempo.
- Problema 7: Igual ao problema 2, com duração das viagens restrita a 160 unidades de tempo.
- Problema 8: Igual ao problema 3, com duração das viagens restrita a 230 unidades de tempo.
- Problema 9: Igual ao problema 4, com duração das viagens restrita a 200 unidades de tempo.
- Problema 10: Igual ao problema 5, com duração das viagens restrita a 200 unidades de tempo.

Os problemas diferem do problema-padrão adotado para a presente dissertação, proposto no capítulo 3, pelo fato de disporem de um único tipo de veículo para o atendimento dos pontos de demanda, apesar de serem submetidos às mesmas restrições de capacidade dos veículos e de duração máxima de operação. Dessa forma, os problemas são de roteamento de veículos, e não de dimensionamento de frota. Os algoritmos não necessitam incorporar um critério de decisão entre diversos tipos de veículos diferentes.

A tabela 7.4 mostra uma comparação entre resultados obtidos pelos algoritmos mencionados. Os custos das soluções através dos algoritmos de Clarke & Wright, varredura e Fisher & Jaikumar foram obtidos do trabalho de Fisher & Jaikumar [11]. Os custos para os demais algoritmos foram obtidos por implementações computacionais dos mesmos desenvolvidas ao longo da dissertação.

TABELA 7.4 - QUADRO COMPARATIVO DE SOLUÇÕES

Problema	C&W	G & M	F&J	PMRG	DF	Dif %	AF	Dif %
1	585	532	524	619	619	18.1%	679	29.6%
2	900	874	857	942	942	9.9%	1051	22.6%
3	886	851	833	965	965	15.8%	1285	54.3%
4	1204	1079	1014	1183	1183	16.7%	1471	45.1%
5	1540	1389	1420	1465	1465	5.5%	1859	33.8%
6	619	560	560	690	690	23.2%	725	29.5%
7	976	933	916	1176	1176	28.4%	1255	37.0%
8	973	888	885	1113	1113	25.8%	1156	30.6%
9	1426	1230	1230	1478	1478	20.2%	1494	21.5%
10	1800	1518	1518	1707	1707	12.5%	1693	11.5%

onde : C&W - Algoritmo de Clarke & Wright;
G&M - Algoritmo de varredura;
F&J - Algoritmo de Fisher & Jaikumar;
PMRG- Algoritmo de partição múltipla de roteiro gigante;
DF - Algoritmo de dimensionamento de frota;
AF - Algoritmo de alocação de frota.

Como pode-se observar, os algoritmos PMRG, DF e AF apresentam os piores resultados médios, para os dez problemas, dentre os seis algoritmos comparados.

Entretanto, os problemas 1 a 5 são problemas básicos de roteamento de múltiplos veículos, tal como definido no capítulo 3, e os problemas 6 a 10 são os mesmos problemas 1 a 5, com adição de uma restrição de duração máxima para as viagens obtidas [11]. O melhor desempenho dos algoritmos C&W, e, principalmente, G&M e F&J se deve ao fato desses algoritmos terem sido formulados para problemas com características similares às do problema básico de roteamento de múltiplos veículos, enquanto o algoritmo de PMRG foi formulado para resolver um problema mais complexo, o problema de tamanho e composição de frota, no qual o enfoque é a obtenção de uma composição de frota, não necessáriamente uniforme, mais econômica, e os algoritmos DF e AF foram formulados para resolver o problema geral de roteamento de múltiplos veículos. Em se tratando de heurísticas, que não necessáriamente encontram a solução ótima, o enfoque de um algoritmo para um dado tipo de problema pode levar, (como no caso), a um pior desempenho resolvendo outros tipos de problemas, se comparado a algoritmos específicos para o mesmo.

Algoritmos desenvolvidos para problemas específicos, por outro lado, são de difícil adaptação para problemas mais abrangentes, como é o caso do problema geral de roteamento de múltiplos veículos. O algoritmo F&J, por exemplo, de melhor qualidade de solução final dentre os algoritmos da tabela 7.4, consegue lidar com restrições de duração máxima de jornada apenas de forma aproximada. Fisher & Jaikumar [11], por exemplo, utilizaram uma aproximação linear para o tempo de viagem gasto no atendimento dos pontos de demanda de cada agrupamento, conseguindo assim restrições aproximadas de duração de viagem, para resolver os problemas 6 a 10 da tabela 7.4, sendo necessária uma checagem final para constatar se realmente nenhuma viagem excede a duração máxima especificada, pois o método não oferece garantias de que essa restrição não seja violada.

Também no trabalho de Brinati et alii [3], a verificação de que nenhuma viagem excede a duração máxima especificada é feita a posteriori, quando utilizado o algoritmo F&J.

Também não foi possível conceber-se uma metodologia que, incorporada ao algoritmo F&J, permitisse selecionar-se a composição de frota mais econômica de frota, a exemplo do algoritmo PMRG. Uma possibilidade alternativa, sugerida por Golden et alii [16], seria de, fixando-se o número total de veículos, obter-se uma composição de frota, aplicando-se depois o algoritmo F&J para obter a solução. Como, a priori, não se conhece o número total de veículos que serão necessários, nem se tem garantias de que a composição de frota obtida seja viável para o problema, ou que propicie uma solução de qualidade razoável, o método não é efetivo para o problema geral de roteamento de múltiplos veículos.

Em suma, longe de desmerecer os algoritmos desenvolvidos na presente dissertação, o resultado obtido na tabela 7.4 ilustra que para problemas particulares deve-se preferencialmente utilizar algoritmos desenvolvidos especificamente para os mesmos. Também a tabela 7.4 mostra que o problema básico de roteamento de múltiplos veículos e o problema de tamanho e composição (para o qual o algoritmo PMRG foi desenvolvido) são casos particulares do problema geral de roteamento de múltiplos veículos. Observa-se, também, que não há diferença nas soluções obtidas pelo algoritmo de partição de roteiro gigante e pelo algoritmo de dimensionamento de frota, o que é fácilmente compreensível devido ao fato de que um deriva do outro, sendo a diferença entre os mesmos (fator redutor de custo fixo) inoperante quando existe um único tipo de veículo disponível.

7.3- SEGUNDA SÉRIE DE TESTES: CENÁRIO RESTRITO

7.3.1- INTRODUÇÃO

Será apresentado agora o comportamento dos algoritmos de dimensionamento e de alocação de frota com a variação de parâmetros do cenário-padrão proposto no capítulo 3.

Os parâmetros variados são, caeteris paribus:

a) Número de pontos de demanda (N);

b) Demanda média dos pontos de demanda (μ_D);

c) Desvio padrão da demanda dos pontos de demanda (σ_D);

d) Distância da base ao centro geométrico dos pontos de demanda (μd_0);

e) Duração da jornada (TV);

f) Distância média dos pontos de demanda ao seu centro geométrico (μd_+);

Conseguiu-se, com esse procedimento, determinar como cada parâmetro influi no comportamento da solução obtida, e no desempenho computacional do algoritmo. Além disso, os testes de comportamento dos algoritmos, por serem realizados em um cenário completo, servem também como comparação das soluções dos algoritmos de dimensionamento e alocação de frota em problemas para os quais foram especificamente desenvolvidos.

Os tipos de veículos disponíveis para a composição da frota foram obtidos do trablho de Brinati et alii [3]. São estimativas de custo fixo

e variável de embarcações de transporte de passageiros do tipo Swath parametrizadas por sua capacidade de passageiros e seu custo, tendo esses veículos sido escolhidos por familiaridade com o trabalho desenvolvido e pelo fato dos mesmos se moverem em um meio bidimensional, como o considerado nos problemas gerados. Os 9 tipos diferentes de veículos considerados são apresentados na tabela 7.5.

TABELA 7.5 - TIPOS DE VEÍCULOS DISPONÍVEIS

Capacidade (unidades)	Velocidade (km/h)	Custo	
		Fixo (Us$)	Variável (Us$/km)
200	33	$769.49	$1.56
200	41	$917.71	$1.73
200	46	$1,108.29	$2.12
250	33	$964.38	$1.95
250	41	$1,156.25	$2.22
250	46	$1,333.16	$2.54
300	33	$1,184.87	$2.44
300	41	$1,421.07	$2.79
300	46	$1,597.00	$3.07

Para a geração dos problemas utilizados nos testes da presente série, baseados na modelagem do cenário-padrao, descrita no capítulo 3, utlizou-se o programa gerador de problemas aleatórios, descrito no apêndice.

A seguir estão representados os valores iniciais de cada parâmetro, estando entre colchetes a sua faixa de variação. Os valores foram escolhidos considerando-se as velocidades e capacidades dos tipos de veículos disponíveis.

item	média	faixa de variação
- N	20	[10,15,20,25,30]
- μ_D	125	[60,125,190,255,320]
- σ_D	20	[0,10,20,30,40]
- μd_0	0	{0,R/2.√2,R.√2,3/2.R.√2,2.R.√2}
- TV	22	[20,21,22,23,24]
- μd_t	70	[50,60,70,80,90]

A seguir será descrito o comportamento dos algoritmos de dimensionamento e de alocação de frota na resolução de problemas aleatórios restritos. Foi criado um programa gerenciador, descrito no apêndice, encarregado de gerar problemas aleatórios e de resolvê-los utilizando os dois algoritmos desenvolvidos, armazenando as soluções em arquivos. A aleatoridade foi conseguida entrando-se com raízes diferentes no gerador de números aleatórios para cada parâmetro variado, apresentados a seguir:

- variação de N	3221747;
- variação de μ_D	5891066;
- variaçao de σ_D	160466;
- variação de μd_0	16151759;
- variaçao de TV	178446;
- variaçao de μd_t	0789;

Nas tabelas 7.6 a 7.17 o tempo de processamento é expresso em centésimos de segundo, o custo total em dólares, a capacidade em unidades de carga e a velocidade em km/hora.

Por último, cabe lembrar que a frota é imposta para o algoritmo de alocação de frota, que eventualmente pode acrescer a mesma de veículos adicionais quando não conseguir alocar todos os pontos de demanda. Por isso, não há sentido em se analisar o comportamento da velocidade e da capacidade médias da frota no algoritmo de alocação de frota, sendo a frota fornecida próxima à frota utilizada (o algoritmo não determina a composição da frota). Por isso, a influência da variação de parâmetros na composição da frota será analisada para o algoritmo de dimensionamento de frota.

Adotou-se, para os testes computacionais, a seguinte notação:

NV:	Número de veículos obtidos/fornecidos;
N_{VG}:	Número de viagens;
CT:	Custo total da solução obtida;
TP:	Tempo de processamento;
μC:	Capacidade média da frota;
μV:	Velocidade média da frota;

7.3.2- VARIAÇÃO DO NUMERO DE PONTOS DE DEMANDA

Observa-se o comportamento das soluções obtidas em 100 problemas aleatórios resolvidos pelo programa de dimensionamento de frota apresentado na tabela 7.6.

TABELA 7.6 - VARIAÇÃO DO NÚMERO DE PONTOS DE DEMANDA (Algoritmo de Dimensionamento de Frota)

N	NV	N_{VG}	C_T	TP	μ_C	μ_V
10	2.00	7.70	3188	133.45	231.25	33.40
15	2.80	11.30	4529	177.15	229.79	33.20
20	3.35	15.05	5701	245.20	233.33	33.90
25	4.05	17.05	7081	306.95	240.75	33.80
30	4.95	21.75	8861	392.85	238.50	33.64

- O numero de veículos, de viagens e o custo da solução aumenta com o aumento do número de pontos de demanda;
- O tempo de processamento tambem aumenta;
- A composição da frota (velocidade e capacidade média) independe da alteração do número de pontos de demanda;

O mesmo observa-se com o algoritmo de alocação de frota, na tabela 7.7.

TABELA 7.7 - VARIAÇÃO DO NÚMERO DE PONTOS DE DEMANDA (Algoritmo de Alocação de Frota)

N	NV	N_{VG}	C_T	TP	μ_C	μ_V
10	2.00	7.45	3285	57.30	231.25	33.40
15	3.05	11.10	4907	166.20	227.29	33.27
20	3.70	14.30	6231	268.25	231.46	33.93
25	4.25	16.75	7607	329.15	238.87	33.78
30	5.55	21.35	9680	540.35	235.08	33.69

Aumentar o número de pontos de demanda não implica em alteração da geometria do cenário do problema, e sim aumentar a complexidade do problema. A conseqüência imediata é um aumento no tempo de processamento, pois ambos os algoritmos terão que considerar um maior

número de pontos de demanda. Também o custo total da solução, o número de viagens e o número de veículos aumentarão porque há mais pontos de demanda. Entretanto, a geometria do problema é a mesma, portanto a composição média da frota (a velocidade e capacidade média) permanece inalterada.

7.3.3- VARIAÇÃO DA DEMANDA MÉDIA

Foram gerados 100 prolemas, nos quais foi variada a demanda média dos pontos de demanda. Os resultados obtidos com o programa de dimensionamento de frota são apresentados na tabela 7.8 (dimensionamento de frota) e 7.9 (alocação de frota), nas quais foi introduzida a capacidade real da frota, $N_{VG} \cdot \mu_C$.

TABELA 7.8 - VARIAÇÃO DA DEMANDA MÉDIA
(Algoritmo de Dimensionamento de Frota)

μ_D	NV	N_{VG}	C_T	TP	μ_C	$N_{VG} \cdot \mu_C$	μ_V
60	2.10	7.30	3465	482.1	213.75	1560.38	34.60
125	3.25	14.85	5757	238.4	236.04	3505.19	34.35
190	4.70	20.00	7547	212.8	219.25	4385.00	34.44
255	5.00	20.10	10559	193.4	282.08	5669.81	35.38
320	7.60	35.70	12582	1241.2	218.87	7813.66	34.32

TABELA 7.9 - VARIAÇÃO DA DEMANDA MÉDIA
(Algoritmo de Alocação de Frota)

μ_D	NV	N_{VG}	C_T	TP	μ_C	$N_{VG} \cdot \mu_C$	μ_V
60	2.30	6.95	3842	206.80	212.50	1476.88	34.60
125	3.65	14.15	6248	274.95	232.50	3289.88	34.22
190	5.10	20.30	7806	281.85	220.83	4482.85	34.47
255	5.10	20.15	10587	215.00	282.45	5691.37	35.36
320	7.75	35.50	12643	534.15	219.21	7781.96	34.35

Observa-se que ocorre uma quebra da tendência observada na composição da frota e no tempo de processamento quando a demanda média é de 320 unidades de carga. Isso ocorre pela ação do procedimento de criação de nós fictícios, pois o tipo de veículo de maior capacidade comporta apenas 300 unidades de carga, o que resulta em aumento do número de viagens realizadas e do tempo de processamento, além de redução da capacidade média dos veículos, sendo o efeito análogo a resolução de um problema de menor demanda média, porem com maior número de pontos de demanda (devido aos nós fictícios).

Para demanda média de até 255 unidades de carga, observa-se o seguinte comportamento:

- Aumento do número de veículos e de viagens realizadas;
- Aumento do custo da solução;
- Não há relação direta com a capacidade média da frota.
- Diminição do tempo de processamento no algoritmo de dimensionamento de frota. No algoritmo de alocação de frota o comportamento é errático;
- A velocidade média da frota apenas aumenta quando a demanda média é elevada, comparada à capacidade dos tipos de veículos disponíveis;

O comportamento da capacidade média da frota não depende únicamente do aumento da demanda. Com o aumento da demanda média deve ocorrer um aumento proporcional na capacidade real da frota (não necessáriamente na sua capacidade média), sendo uma diminuição da capacidade média da frota compensada por um aumento do número de viagens. Dessa forma, a capacidade média da frota é função também do número de viagens realizadas.

Também a velocidade média da frota não apresenta uma relação direta com o aumento da demanda média da frota. Observa-se um aumento da velocidade quando a demanda média é próxima da capacidade do maior tipo de veículo disponível, sendo a única alteração na composição da frota possível como resposta a um aumento da demanda média.

A diminuição do tempo de processamento com o aumento da demanda média no algoritmo de dimensionamento de frota ocorre devido à diminuição do número de partições possíveis do roteiro gigante, causada pela diminuição do número de pontos de demanda que podem ser alocados em cada viagem (ação da restrição de capacidade).

Por último, os aumentos de custo e do número de veículos são decorrência do aumento da demanda média da frota.

7.3.4- VARIAÇÃO DO DESVIO PADRÃO DA DEMANDA

Variou-se o desvio parão da demanda em torno da média de 125 unidades em 100 problemas, cujos resultados são reproduzidos na tabela 7.10a (programa de dimensionamento de frota) e na tabela 7.11a (programa de alocação de frota).

TABELA 7.10a - VARIAÇÃO DO DESVIO PADRÃO DA DEMANDA (Algoritmo de Dimensionamento de Frota)

σ_D	NV	N_{VG}	C_T	TP	μ_C	N_{VG}-μ_C	μ_V
0	3.10	12.80	5409	229.90	242.08	3098.62	33.67
10	3.30	14.25	6096	233.05	246.04	3506.07	34.03
20	3.30	14.55	5781	236.10	240.21	3495.06	33.60
30	3.35	14.55	5828	222.70	235.42	3425.36	33.87
40	3.50	14.55	6024	240.25	228.75	3328.31	34.23

TABELA 7.11a - VARIAÇÃO DO DESVIO PADRÃO DA DEMANDA (Algoritmo de Alocação de Frota)

σ_D	NV	N_{VG}	C_T	TP	μ_C	$N_{VG} \cdot \mu_C$	μ_V
0	3.20	11.95	5540	187.95	241.04	2880.43	33.67
10	3.60	13.80	6518	239.90	243.54	3360.85	34.00
20	3.60	14.25	6208	254.25	236.25	3366.56	33.67
30	3.75	14.65	6362	274.85	232.25	3402.46	33.83
40	3.85	14.30	6482	256.95	227.71	3256.25	34.23

Analisando-se a capacidade real da frota, observa-se que ocorre um grande aumento da mesma quando o desvio padrão da demanda é não nulo. Isso ocorre porque a demanda média de 125 unidades é metade de 250, capacidade de um dos tipos de veículos disponíveis. Com desvio padrão nulo, esse tipo de veículo teve potencial para realizar muitas viagens com 100% de ocupação, diminuindo o número de veículos e viagens necessários, e, conseqüentemente, o custo.

Realizaram-se novos testes, utilizando-se, porém, demanda média de 126 unidades, com os demais parâmetros inalterados, obtendo-se os resultados das tabelas 7.10b e 7.11b.

TABELA 7.10b - VARIAÇÃO DO DESVIO PADRÃO DA DEMANDA (Algoritmo de Dimensionamento de Frota)

σ_D	NV	N_{VG}	C_T	TP	μ_C	$N_{VG} \cdot \mu_C$	μ_V
0	3.7	17.1	6448	240.82	235.83	4032.69	33.5
10	3.35	14.5	5913	226.4	240.21	3483.05	33.27
20	3.25	14.7	5827	229.57	240.63	3537.26	34.07
30	3.4	14.2	5782	221.79	232.5	3301.5	33.73
40	3.45	14.9	6081	243.42	231.25	3445.63	34.88

TABELA 7.11b - VARIAÇÃO DO DESVIO PADRÃO DA DEMANDA (Algoritmo de Alocação de Frota)

σ_D	NV	N_{VG}	C_T	TP	μ_C	N_{VG}-μ_C	μ_V
0	3.75	16.85	6480	134.24	235.42	3966.83	33.47
10	3.9	14.3	6612	303.14	235.63	3369.51	33.51
20	3.65	14.3	6314	249.16	236.88	3387.38	34.17
30	3.8	14.00	6343	246.14	228.54	3199.56	33.77
40	3.8	13.75	6501	227.53	228.67	3144.21	34.73

Observa-se que para padrão da demanda elevados aproximam os resultados obtidos com demanda média de 125 e 126 unidades, enquanto que para desvio padrão nulo os resultados obtidos para essas demandas são diferentes. Dessa forma, um desvio padrão elevado, cuja ordem de grandeza seja bem maior que a da alteração da demanda média, "amortece" o efeito dessa alteração.

Um desvio padrão pequeno (se comparado com a demanda média) implica em uma relação muito maior da solução obtida com a demanda média, para um dado conjunto de tipos de veículos, que desvios padrão maiores. A variação do desvio padrão, entretanto, por si só não apresenta influência direta no comportamento dos parâmetros estudados, não podendo ser feita a correlação dos mesmos com um aumento ou diminuição do desvio padrão da demanda.

7.3.5- VARIAÇÃO DA POSIÇÃO RELATIVA DA BASE AO CENTRO GEOMÉTRICO DOS PONTOS DE DEMANDA

Com o objetivo de testar a influência da geometria dos problemas variou-se a posição relativa da base em 100 problemas resolvidos pelos programas de dimensionamento e de alocação de frota, com resultados reproduzidos nas tabelas 7.12 e 7.13, respectivamente.

TABELA 7.12- VARIAÇÃO DA POSIÇÃO RELATIVA DA BASE (Algoritmo de Dimensionamento de Frota)

μd_0	NV	N_{VG}	C_T	TP	μ_C	μ_V
0	3.35	14.50	5732	232.80	236.04	33.27
$R/2\sqrt{2}$	3.95	12.90	7492	234.20	248.96	33.73
$R\sqrt{2}$	5.05	11.85	10680	226.95	253.42	34.52
$3R/2\sqrt{2}$	5.95	11.30	14619	234.20	257.83	37.09
$2.R.\sqrt{2}$	7.40	10.80	19364	270.75	262.11	37.84

TABELA 7.13- VARIAÇÃO DA POSIÇÃO RELATIVA DA BASE (Algoritmo de Alocação de Frota)

μd_0	NV	N_{VG}	C_T	TP	μ_C	μ_V
0	3.70	14.00	6241	262.50	233.17	33.33
$R/2\sqrt{2}$	4.15	13.05	8000	244.15	247.83	33.69
$R\sqrt{2}$	5.60	12.65	11538	331.05	249.01	34.82
$3R/2\sqrt{2}$	6.80	12.40	16066	447.60	252.86	37.22
$2.R.\sqrt{2}$	8.90	11.85	21868	545.45	258.15	37.17

Em ambos os programas constatou-se

- Aumento do número de veículos, do custo da solução e da capacidade e velocidades médias da frota;
- Diminuição do número de viagens de cada veículo;
- Tempo de processamento constante no programa de dimensionamento de frota, e crescente no de alocação de frota;

Quanto mais afastada a base do conjunto de pontos de demanda, tanto maior o tempo de viagem dos veículos para deslocarem-se da mesma ao seu "local de trabalho". Por isso, o algoritmo de dimensionamento de frota procura maximizar o tempo gasto atendendo os nós, diminuindo a parcela do tempo em viagem do tempo total de operação. Faz isso aumentando a

capacidade média (podem atender a mais nós sem retornar à base) e a velocidade média da frota (gastam menos tempo se movimentando).

A diminuição do número de viagens é função do maior afastamento da base, sendo que o aumento da velocidade média procura compensar em parte isso.

O tempo de processamento é práticamente constante no programa de dimensionamento de frota (exceto quando a base é muito distante dos pontos de demanda). Já no programa de alocação de frota o tempo de processamento cresce sensivelmente com o afastamento da base do centro geométrico dos pontos de demanda, função direta do aumento do número de veículos e, conseqüentemente, do aumento do número de execuções do procedimento de alocação de pontos de demanda.

7.3.6- VARIAÇÃO DA DURAÇÃO MÁXIMA DE OPERAÇÃO DOS VEÍCULOS

Em 100 problemas aleatórios obteve-se os resultados expressos na tabela 7.14, para o problema de dimensionamento de frota, e 7.15, para o problema de alocação de frota.

TABELA 7.14- VARIAÇÃO DA DURAÇÃO MÁXIMA DE OPERAÇÃO (Algoritmo de Dimensionamento de Frota)

TV	NV	N_{VG}	C_T	TP	μ_C	μ_V
20	3.40	13.95	6123	228.20	243.33	35.37
21	3.60	15.20	6106	235.30	234.79	34.02
22	3.55	14.85	5962	234.20	233.75	33.40
23	3.15	14.45	5581	235.15	235.63	33.67
24	3.20	14.35	5590	238.25	236.25	33.10

TABELA 7.15- VARIAÇÃO DA DURAÇÃO MÁXIMA DE OPERAÇÃO (Algoritmo de Alocação de Frota)

TV	NV	N_{VG}	C_T	TP	μ_C	μ_V
20	3.85	13.90	6842	293.00	240.00	35.00
21	3.85	14.80	6584	245.70	232.50	33.92
22	3.60	14.60	6369	193.75	233.13	33.40
23	3.50	13.80	6044	271.10	233.08	33.63
24	3.40	13.80	6006	227.45	234.17	33.10

- A variação da duração máxima de operação influe diretamente na velocidade média da frota, e, conseqüentemente no custo. Ambos diminuem à medida que se aumenta a duração máxima de operação dos veículos;
- O número de veículos da frota também diminue com o aumento da duração máxima da operação;
- A influência da variação da duração máxima de operação no número de viagens e na capacidade média da frota não é direta;
- O tempo de processamento aumenta no programa de dimensionamento de frota com o aumento da duração máxima de operação, e o tempo de processamento do programa de alocação de frota não depende da variação da mesma.

Com o aumento da duração máxima permitida para a operação dos veículos não ocorre uma alteração da geometria dos problemas. O comportamento do algoritmo de dimensionamento de frota é diminuir a velocidade média da frota com o aumento da duração máxima permitida de operação, para minimizar custos. Com isso, o custo também cai.

Menos direta é a influência no número de veículos, da capacidade e no número de viagens da frota. Maior duração permitida para as viagens possibilita mais viagens realizadas pelos veículos, o que possibilita

uma menor frota. Também a capacidade média da frota pode ser diminuída, aumentando-se o aproveitamento (número de viagens) dos veículos da frota. Por outro lado, diminuição da velocidade média implica em potencial diminuição do número de viagens. Com isso, a alteração da capacidade média, do número de veículos e do número de viagens depende da conjugação do aumento da duração máxima permitida e da diminuição da velocidade. Como diminuir número de veículos implica em diminuir o custo fixo, nota-se diminuição do mesmo, enquanto que o comportamento do número de viagens e da capacidade média da frota não é tão direto.

Finalmente, o tempo de processamento do algoritmo de dimensionamento de frota aumenta com o aumento da duração máxima permitida de operação. Isso porque aumenta o número de partições possíveis do roteiro gigante. Já o tempo de processamento do algoritmo de alocação de frota não depende diretamente da alteração da duração máxima de operação.

7.3.7- VARIAÇÃO DA DISTÂNCIA MÉDIA DOS PONTOS DE DEMANDA AO SEU CENTRO GEOMÉTRICO

Variou-se a distância média dos pontos de demanda ao seu centro geométrico em 100 problemas aleatórios, resolvidos pelo programa de dimensionamento (tabela 7.16) e de alocação de frota (tabela 7.17).

TABELA 7.16- VARIAÇÃO DA DISTÂNCIA MÉDIA (Algoritmo de Dimensionamento de Frota)

μd_+	NV	N_{VG}	C_T	TP	μ_C	μ_V
50	2.85	15.10	4611	223.35	232.92	34.00
60	3.00	14.75	5153	229.95	240.00	33.67
70	3.50	14.65	6161	236.90	238.33	33.93
80	3.60	14.55	6394	237.05	238.75	33.80
90	3.90	14.25	7228	233.85	242.12	33.73

TABELA 7.17- VARIAÇÃO DA DISTÂNCIA MÉDIA (Algoritmo de Alocação de Frota)

μd_i	NV	N_{VC}	C_T	TP	μ_C	μ_V
50	3.00	14.60	4878	203.35	231.67	33.93
60	3.25	14.10	5563	228.45	237.92	33.67
70	3.85	14.35	6680	259.90	234.79	33.90
80	3.85	14.10	7013	237.20	237.29	33.73
90	4.50	14.20	8060	244.95	236.54	33.92

- O número de veículos da frota aumenta, com o aumento da distância, à medida que diminue o número de viagens realizadas. O custo total aumenta em função do aumento do número de veículos;
- Não observou-se tendência definida de alteração da velocidade média da frota e da capacidade média da mesma;
- O comportamento do tempo de processamento é errático para o algoritmo de dimensionamento de frota. Já o tempo de processamento do algoritmo de alocação de frota é independente da alteração da distância média dos pontos de demanda ao seu centro geométrico;

Ocorre uma relação direta do aumento do número de veículos da frota com o aumento da distância média dos pontos de demanda ao seu centro geométrico. Com a escala de aumento escolhida, (variação de 10 em 10 km, com a área adicional do cenário variando πR^2), aumentos na capacidade e na velocidade médias da frota, (a maneira que o algoritmo optou quando foi variada a posição da base, um problema em conceito igual ao do afastamento dos pontos de demanda de seu centro geométrico), com os tipos de veículos disponíveis, seriam insuficientes. Por isso, foi necessário o aumento do número de veículos da frota. Conseqüentemente, ocorreu diminuição do número de viagens, pois manteve-se o número de pontos de demanda constantes.

Com isso, a alteração na composição média da frota é errática, representando tentativas do algoritmo de dimensionamento de frota de realizar o "ajuste fino" da frota, procurando reduzir o custo variável com o aumento de custo fixo ocorrido. Pode-se interpretar isso a uma oscilação de segunda ordem, face a uma oscilação de primeira ordem (aumento do número de veículos).

Por último, o tempo de processamento do algoritmo de alocação de frota não apresentou relação direta com a alteração da distância média entre os pontos de demanda e seu centro geométrico. A tendência do comportamento do tempo de processamento do algoritmo de dimensionamento de frota deveria ser de diminuição com o aumento da distância entre os pontos de demanda e o seu centro geométrico (menor número de partições viáveis possíveis), entretanto observou-se o oposto na maioria dos casos. A causa disso é um aumento do número de viagens construídas com o fator redutor de custo fixo e que não podem ser combinadas, resultando em um aumento da duração do pós-processamento da solução.

7.3.8- MELHORIA DA SOLUÇÃO OBTIDA

Nos testes computacionais realizados em cenário restrito observou-se que ambos os algoritmos apresentam comportamento aderente dos resultados obtidos com variação dos parâmetros, (sem levar-se em consideração a composição média da frota, que é fornecida pelo algoritmo de dimensionamento de frota, portanto obrigatóriamente deve ser aderente), à exceção do tempo de processamento, que depende primeiramente da formulação do algoritmo. Dessa forma, se, perante um dado estímulo, o custo obtido pelo programa de dimensionamento de frota aumentar, ocorrerá um aumento também no custo obtido pelo programa de alocação de frota. Entretanto, pelo fato do funcionamento dos dois algoritmos diferir, a magnitude da alteração pode ser diferente.

O algoritmo de melhoria de soluções, cuja implementação computacional está no apêndice, foi desenvolvido para diminuir o custo variável das soluções obtidas pelos algoritmos de dimensionamento e de alocação de frota, tendo capacidade para diminuir também o seu custo fixo, conforme exposto no capítulo 6. Essa implementação revelou-se de processamento demorado, o que impediu que fosse realizada uma série de testes extensiva como nos dois outros algoritmos.

Foram realizados oito testes em um computador IBM-AT, com características expressas na tabela 7.18, onde:

- Random é a semente do gerador de números aleatórios adotada;
- N - Número de pontos de demanda;
- μd_+ - Distância média dos pontos de demanda ao seu centro geométrico;
- μ_D - Demanda média dos pontos de demanda;
- σ_D - Desvio padrão da demanda dos pontos de demanda;
- μd_0 - Distância da base ao centro geométrico dos pontos de demanda;
- mTC - Média da taxa de carregamento dos pontos de demanda;
- sTC - Desvio padrão da taxa de carregamento dos pontos de demanda;
- TV - Duração da jornada;

TABELA 7.18 - PROBLEMAS QUE TIVERAM SOLUÇÃO MELHORADA

	Random	N	μd_1	μD	σD	μTC	σTC	μd_0	TV
1	5028	20	60	200	30	250	5	127.3	23
2	1259	20	90	100	20	250	5	0	21
3	6122	20	70	250	50	250	5	99.0	20
4	3551	20	70	150	20	250	5	99.0	24
5	6869	20	40	50	5	250	5	28.3	20
6	2529	20	80	260	0	250	5	0	21
7	26	20	50	50	10	250	5	35.4	19
8	864	20	40	230	30	250	5	0	15

Os tipos de veículos disponíveis são apresentados na tabela 7.19.

TABELA 7.19 - TIPOS DE VEICULOS DISPONÍVEIS

Capacidade (unidades)	Velocidade (km/h)	Custo	
		Fixo (Us$)	Variável (Us$/km)
200	33	$769.49	$1.56
200	41	$917.71	$1.73
200	46	$1,108.29	$2.12
250	33	$964.38	$1.95
250	41	$1,156.25	$2.22
250	46	$1,333.16	$2.54

Os resultados são mostrados nas tabelas 7.20 a 7.22. A tabela 7.20 ilustra o desempenho do algoritmo de melhoria em soluções geradas pelo algoritmo de dimensionamento de frota. A tabela 7.21 o faz com soluções geradas pelo algoritmo de alocação de frota, tendo a frota sido gerada pelo algoritmo de dimensionamento de frota, diferentemente da tabela 7.22, na qual a solução e a composição da frota são geradas pelo algoritmo de alocação de frota. Os tempos de processamento (T.Proc.) são expressos em (minutos:segundos:centésimos) e os custos totais da solução em dólares. N.Veic. indica o número de veículos que compõe a frota.

TABELA 7.20 - MELHORIA DA SOLUÇÃO DO ALGORITMO DE DIMENSIONAMENTO DE FROTA

Problema	SOLUÇÃO INICIAL			SOLUÇÃO MELHORADA		
	Custo	N.Veic.	T.Proc.	Custo	N.Veic.	T.Proc.
1	19096	10	0:39	19085	10	15:98
2	5858	3	0:33	5858	3	5:88
3	21790	12	0:77	21505	12	1:17:16
4	11978	7	0:39	11978	7	12:41
5	2701	2	0:88	2639	2	11:59
6	13095	8	1:64	11094	7	14:27:16
7	3036	2	0:66	2959	2	14:39
8	7633	5	0:55	7574	5	44:59

TABELA 7.21 - MELHORIA DA SOLUÇÃO DO ALGORITMO DE ALOCAÇÃO DE FROTA (FROTA FORNECIDA)

Problema	SOLUÇÃO INICIAL			SOLUÇÃO MELHORADA		
	Custo	N.Veic.	T.Proc.	Custo	N.Veic.	T.Proc.
1	18994	10	0:44	18994	10	12:03
2	7184	4	1:04	6614	4	32:57
3	19593	11	0:88	19095	11	1:41:22
4	11978	7	0:28	11978	7	12:24
5	2725	2	0:16	2659	2	8:29
6	11101	7	0:88	11101	7	58:76
7	3267	2	0:16	3104	2	13:84
8	7463	5	0:38	7429	5	46:62

TABELA 7.22 - MELHORIA DA SOLUÇÃO DO ALGORITMO DE ALOCAÇÃO DE FROTA (FROTA GERADA PELO PRÓPRIO ALGORITMO)

Problema	SOLUÇÃO INICIAL			SOLUÇÃO MELHORADA		
	Custo	N.Veic.	T.Proc.	Custo	N.Veic.	T.Proc.
1	20728	10	1:05	20728	10	12:78
2	7067	4	0:55	6891	4	8:01
3	20794	11	1:27	20751	11	34:98
4	14283	7	0:83	14266	7	20:81
5	2959	2	0:50	2889	2	8:68
6	12772	7	1:27	12772	7	56:72
7	3491	2	0:50	3237	2	13:57
8	7124	5	0:88	7084	5	20:54

Nos problemas 3, 6 e 8 os algoritmos geraram pontos de demanda fictícios (em número de 9, 20 e 3, respectivamente), por estarem disponíveis veículos com capacidade máxima de 250 unidades, o que foi insuficiente, nesses problemas, para atender a todos os pontos de demanda.

A maior redução observada no custo total foi de 15.3%, no problema 6, resolvido pelo algoritmo de dimensionamento de frota, quando ocorreu também uma redução do custo fixo (a frota passou de 8 para 7 unidades). Houve redução do custo variável de boa parte dos problemas, mostrando que o algoritmo é efetivo nessa tarefa. Entretanto, dada a natureza iterativa do algoritmo de melhoria de soluções, fortemente dependente da solução inicial, não é possível determinar com antecedência se ocorrerá uma redução sensível do custo variável de uma solução com a sua aplicação, ou qual será a duração do processamento.

Por último, cabe observar que o algoritmo de alocação de frota tem capacidade de dimensionamento de frota, sendo a mesma obtida pelo critério de ordenação de veículos, sendo conveniente ressaltar que o mesmo é míope, ao contrário do critério de seleção do algoritmo de dimensionamento de frota, o que foi responsável por frotas com maior custo total, quando obtidas pelo algoritmo de alocação de frota.

7.4- CONSIDERAÇÕES FINAIS

As tabelas 7.23 e 7.24 são um resumo do comportamento dos algoritmos de dimensionamento e de alocação de frota com o aumento de parâmetros dos problemas, que adota a seguinte simbologia:

⇑	Aumento com o aumento do parâmetro considerado;
⇓	Diminuição com o aumento do parâmetro considerado;
⊘	O item independe, ou não depende exclusivamente, do parâmetro considerado;

TABELA 7.23- COMPORTAMENTO DO ALGORITMO DE DIMENSIONAMENTO DE FROTA

	NV	N_{VG}	C_T	TP	μ_C	μ_V
N	⇑	⇑	⇑	⇑	⊗	⊗
μD	⇑	⇑	⇑	⇓	⊗	⊗
σD	⊗	⊗	⊗	⊗	⊗	⊗
μd_o	⇑	⇓	⇑	⊗	⇑	⇑
TV	⇓	⊗	⇓	⇑	⊗	⇓
μd_+	⇑	⇓	⇑	⊗	⊗	⊗

TABELA 7.24- COMPORTAMENTO DO ALGORITMO DE ALOCAÇÃO DE FROTA

	NV	N_{VG}	C_T	TP
N	⇑	⇑	⇑	⇑
μD	⇑	⇑	⇑	⊗
σD	⊗	⊗	⊗	⊗
μd_o	⇑	⇓	⇑	⇑
TV	⇓	⊗	⇓	⊗
μd_+	⇑	⇓	⇑	⊗

Ambos os algoritmos desempenham a tarefa para a qual foram desenvolvidos, o dimensionamento e alocação de uma frota de veículos

para o atendimento de um conjunto de pontos de demanda, ou a alocação de uma frota fornecida de veículos. Conseguem resolver problemas levando em consideração custos fixos e operacionais dos veículos, consideram restrições de duração máxima de operação e de capacidade de carga, além de poderem facilmente incorporar outras restrições. Por último conseguem atender a pontos de grande demanda, que necessitam da alocação de mais de um veículo.

O outro lado da moeda é que os algoritmos desenvolvidos apresentam um desempenho inferior, principalmente quanto à qualidade da solução obtida, quando utilizados para resolver problemas mais simples, como o de roteamento de múltiplos veículos idênticos, se comparados com outros algoritmos especificamente desenvolvidos para esse fim.

O algoritmo de alocação de frota, altamente iterativo, apresenta comportamento semelhante ao de dimensionamento de frota, com a variação de parâmetros. A solução, entretanto, é obtida de maneira diversa à do algoritmo de dimensionamento de frota, podendo diferir na frota necessária para a alocação dos pontos de demanda em problemas restritos.

O algoritmo de melhoria de soluções, também combinatório, é de execução demorada, entretanto, freqüentemente, consegue ter sucesso na obtenção de soluções de custos totais menores que as soluções iniciais fornecidas, principalmente devido à redução dos custo variável, tendo potencial, entretanto, para uma redução também do custo fixo da frota, podendo diminuir o tamanho da mesma.

Com estas considerações, os algoritmos prestam-se para a finalidade almejada, e revelaram-se, com os testes computacionais, rápidos e obtendo soluções aceitáveis, ficando a se determinar o quão distantes estão das ótimas, em média, para o que teria que se resolver ótimamente

diversos problemas com as características definidas no capítulo 3, e compará-los com os algoritmos desenvolvidos, o que foge ao escopo da presente dissertação.

8- CONCLUSÃO

8.1- INTRODUÇÃO

Pelo fato da maioria dos algoritmos para roteamento de múltiplos veículos existentes ter sido desenvolvida para resolver problemas específicos, muitos problemas reais não podem ser resolvidos diretamente pelos mesmos, exigindo grandes alterações estruturais, ou mesmo a formulação de novos algoritmos, devido a diferenças entre os cenários para os quais os algoritmos foram desenvolvidos e o cenário do problema real.

O objetivo dessa dissertação foi a formulação de algoritmos capazes de resolver uma variada gama de problemas reais. Para isso, os mesmos foram baseados em um cenário-padrão muito abrangente, do qual muitos problemas reais fossem simplificações. Com isso, para muitos problemas reais, deixa de ser necessária uma alteração estrutural dos algoritmos desenvolvidos, podendo os mesmos ser aplicados diretamente.

Tambem foi dada ênfase na capacidade de implementação dos algoritmos em microcomputaores. Os algoritmos desenvolvidos são de execução computacional rápida, e tem baixo consumo de memória (durante a fase de desenvolvimento do programa de dimensionamento de frota um problema de 300 pontos de demanda foi resolvido em um microcomputador de 16 bits tipo IBM XT). Com o advento, no Brasil, da linha de microcomputadores de 32 bits em grande escala, é possível resolver rápidamente problemas de grande porte, o que permite, através de variações nos algoritmos, obter diversas soluções diferentes, podendo-se escolher a de menor custo.

Foram obtidos três algoritmos na dissertação, um para o dimensionamento de frota (obtenção de uma frota de veículos, e o seu roteamento,

para atender um conjunto de pontos de demanda), outro para a alocação de frota (utilização de uma frota conhecida para atender um conjunto de pontos de demanda), e um terceiro de melhoria de soluções, obtidas não apenas pelos dois algoritmos, como por qualquer algoritmo, desde que o problema seja baseado no cenário-padrão adotado no presente trabalho.

8.2- CONCLUSÕES

Os objetivos propostos para a presente dissertação foram atingidos. As seguintes conclusões podem ser obtidas:

- O cenário padrão é muito abrangente. Problemas comuns como o básico de roteamento de múltiplos veículos e o de tamanho e composição de frota são simplificações do problema geral de roteamento de múltiplos veículos.
- O desempenho dos algoritmos em problemas menos restritos é inferior no que diz respeito à qualidade de solução obtida, se comparado com soluções obtidas por algoritmos especificamente desenvolvidos para esses problemas, como ficou demonstrado nos testes comparativos do capítulo 7.
- A estratégia de redução artificial do custo fixo no algoritmo de dimensionamento de frota, (utilização do fator redutor de custo fixo), mostrou-se efetiva, corroborando a conclusão de Chih [4] de que a estratégia de utilizar-se uma frota composta de veículos menores e mais rápidos, capazes de realizar mais viagens, é eficaz na redução do custo total.
- Para um dado conjunto de pontos de demanda e uma dada frota, obtida pelo algoritmo de dimensionamento de frota, a solução obtida por esse algoritmo tende a ter custo total menor que o custo total da solução obtida pelo algoritmo de alocação de frota (com exceções).

- O algoritmo de alocação de frota possui também uma capacidade limitada de dimensionamento de frota, sendo a seleção da frota realizada pelo critério de ordenação de veículos. Convém ressaltar que o algoritmo de dimensionamento de frotas tende a obter frotas que resultam em menor custo total que o algoritmo de alocação de frota, entretanto foram constatados casos em que aconteceu o contrário (tabelas 7.20 e 7.22). Sendo os algoritmos de execução computacional rápida recomenda-se que dimensione-se a frota utilizando ambos, variando-se o critério de ordenação de veículos do algoritmo de alocação de frota, e fronecendo-se vários roteiros gigantes diferentes para o algoritmo de dimensionamento de frota.
- O algoritmo de melhoria de soluções, apesar de ter execução computacional demorada, é efetivo na redução do custo variável de soluções fornecidas para problemas baseados no problema geral de roteamento de múltiplos veículos. Também possui capacidade limitada de redução de custo fixo, ocorrendo problemas nos quais o tamanho da frota foi reduzido. Tem a desvantagem, além do tempo de processamento, a impossibilidade de predizer-se qual a melhoria que será proporcionada pela sua utilização, sendo a melhoria fortemente dependente da solução inicial. Recomenda-se a sua utilização, pois no pior dos casos, a solução fornecida não sofrerá alteração, havendo grande potencial de redução de custo.

8.3- RECOMENDAÇÕES PARA NOVOS TRABALHOS

Os testes de desempenho realizados nos algoritmos foram todos comparativos, isto é, verificava-se se a solução obtida por uma heurística era melhor ou pior que a obtida por outra heurística. Não foi possível obter-se o quanto as soluções obtidas pelos algoritmos de dimensionamento e de alocação de frota eram piores que a solução ótima, pelo fato de não dispor-se de soluções ótimas, pela própria natureza do problema que se propôs resolver. Seria interessante, para futuros

trabalhos, a comparação, mesmo que em problemas de pequeno porte, com soluções ótimas, para constatar se os algoritmos podem ser utilizados com garantia de boas soluções, ou se deve-se investir em outras abordagens para resolver problemas de dimensionamento e de alocação de frota do tipo dos adotados nesse trabalho.

Seria interessante o estudo de novas abordagens para a resolução dos problemas de dimensionamento e de alocação de frota. Pode ser interessante aplicar o conceito do fator redutor de custo fixo em outros algoritmos para o problema de tamanho e composição de frota (Golden et alii, [16]), e tentar obter outros algoritmos de dimensionamento de frota para problemas baseados no cenário padrão da presente dissertação. Também seria muito interessante o desenvolvimento de outros algoritmos de alocação de frota, pois a abordagem adotada é fortemente míope, apesar dos esforços em contrário. Na verdade o problema de alocação de frota apresenta um grau de liberdade a mais que o problema de dimensionamento de frota, pois, sendo o número de veículos de cada tipo limitado (a frota é conhecida), o algoritmo deve considerar que a utilização de um veículo em um dado momento implicará na sua indisponibilidade no futuro. Da mesma forma, novos algoritmos de melhoria de soluções constituem um tema interessante para novos trabalhos.

Também pode ser interessante que outros trabalhos aprofundem-se mais nos algoritmos desenvolvidos. Novos critérios de ordenação de veículos e de "permutação" de pontos de demanda, (para minimizar a miopia), podem melhorar o desempenho do algoritmo de alocação de frota. A proposição de outros métodos para lidar com o problema de pontos de demanda com excesso de demanda pode ser interessante. Por último, seria interessante um estudo da influência de diferentes roteiros gigantes na solução do algoritmo de dimensionamento de frota.

O próprio cenário padrão pode ser ampliado. Particularmente a inclusão de mais de uma base, com os veículos podendo iniciar a viagem em uma base e terminar em outra, parece ser um tema muito interessante para novos trabalhos. Também a possibilidade dos veículos de operarem com mais de um tipo de carga pode ser incluída, para tornar o cenário padrão ainda mais abrangente.

Por último, mas não menos importante, a aplicação dos algoritmos desenvolvidos em problemas reais, particularmente de grande porte, e o confronto das soluções obtidas com as soluções adotadas na realidade, poderia mostrar o quanto os algoritmos desenvolvidos podem auxiliar na solução de problemas reais de dimensionamento e alocação de múltiplos veículos.

São Paulo, 12 de outubro de 1992.

BIBLIOGRAFIA

1 **BODIN L.,GOLDEN B.** "Classification in vehicle routing and scheduling", Networks, vol 11, (1981)

2 **BODIN L.,GOLDEN B. ASSAD A.,BALL M.** "Routing and scheduling of vehicles and crews", Computers & Operations Research, vol 10, n.2, (1983)

3 **BRINATI M. ET ALII** "Sistema de transporte de pessoal para as plataformas da bacia de campos", Projeto 206/86 Cepen/USP, (1989)

4 **CHIH, W. Y.** "Influência dos custos fixos e variáveis na roteirização de frotas de veículos de capacidades variadas", Dissertação de Mestrado, Escola Politécnica/USP, (1989)

5 **CHRISTOFIDES N., EILON S.** "An algorithm for the vehicle-dispatching problem", Operational Research Quarterly 20, (1969)

6 **CHRISTOFIDES N., MINGOZZI A., TOTH P.** "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations", Mathematical Programming 20, (1981)

7 **CHRISTOFIDES N., MINGOZZI A., TOTH P.** "State-space relaxation procedures for the computation of bounds to routing problems", Networks, vol 11, (1981)

8 **CHRISTOFIDES N.** "Bounds for the travelling salesman", Operations Research, vol 20, (1972)

9 **CLARKE G.,WRIGHT J.** "Scheduling of vehicles from a central depot to a number of delivery points", Operations Research, vol 12, (1964)

10 **CORNUEJOLS G., FISHER M., NEMHAUSER G.** "Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms", Management Science, vol 23, n.8, (1977)

11 **FISHER M.,JAIKUMAR R.** "A generalized assignment heuristic for vehicle routing", Net-works, vol 11, (1981)

12 **FISHER M.** "The lagrangian relaxation method for solving integer programming problems", vol 27, n.1, (1981)

13 **GILLETT B.,MILLER L.** "A heuristic algorithm for the vehicle-dispatch problem", Operations Research, vol 22, (1974)

14 **GOLDEN B., ASSAD A.** "Perspectives on vehicle routing: exciting new developments", Operations Research, vol 34, n.5, (1986)

15 **GOLDEN B., ASSAD A.** "A decision-theoretic framework for comparing heuristics", European Journal of Operational Research 18, (1984)

16 **GOLDEN B.,ASSAD A., LEVY L.,GHEYSENS F.** "The fleet size and mix routing problem", Management Science & Statistics Working Paper n. 82-020, University of Maryland at College Park, (1982)

17 **GOUVÊA M.** "Adaptação de heurísticas para a resolução do problema de dimensionamento de frota", Seminário de Área, Escola Politécnica/USP, (1989)

18 **GOUVÊA M.** "Implementação de um algoritmo para o dimensionamento de frota". Trabalho de Conclusão da Disciplina "Implementação de Algoritmos de P.O.", Escola Politécnica/USP, (1989)

19 **JAW J.J.** "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows", Transportation Research B, vol 20 B, n. 3, (1986)

20 **LARSON R., ODONI A.** "Urban operations research", pg 416-423, Prentice Hall, Englewood Cliffs, New Jersey, (1980)

21 **LENSTRA J. RINNOOY KAN A.** "Complexity of vehicle routing and scheduling problems", Networks, vol 11, (1981)

22 **MÜLLER-MERBACH, H.** "Heuristics and their design: a survey", European Journal of Operational Research 8, (1981)

23 **PAESSENS H.** "The savings algorithm for the vehicle routing problem", European Journal of Operational Research 34, (1988)

24 **PSARAFTIS N.** "k-Interchange procedures for local search in a precedence-constrained routing problem", European Journal of Operational Research 13, (1983)

25 **RONEN D.** "Perspectives on practical aspects of truck routing and scheduling", European Journal of Operational Research 35, (1988)

26 **SILVER E., VIDAL R., de WERRA D.** "A tutorial on heuristic methods", European Journal of Operational Research 5, (1980)

27 **ZANAKIS S., EVANS J.** "Heuristic 'optimization': why, when, and how to use it", Interfaces, vol 11, n.5, (1981)

APÊNDICE - LISTAGENS DOS PROGRAMAS DE COMPUTADOR

I) PROGRAMA DE DIMENSIONAMENTO DE FROTA

O programa de dimensionamento de frota é a implementação computacional do algoritmo de dimensionamento de frota, exposto no capítulo 4.
Deve ser fornecido um arquivo, de quatro registros reais ,contendo todos os nós do problema a ser resolvido.
A primeira linha deve possuir informações da base da frota, o primeiro registro fornecendo a coordenada X, o segundo a coordenada Y, o quarto a taxa de carregamento da base. O terceiro registro fornece a duração máxima de jornada do problema.
As demais linhas devem possuir informações dos pontos de demanda da frota (uma linha para cada ponto de demanda). O primeiro registro fornece a coordenada X, o segundo a coordenada Y, o terceiro a sua demanda e o quarto a sua taxa de carregamento.
Deve também ser fornecido um arquivo, de quatro registros reais, contendo os tipos de veículos disponíveis, cada linha contendo informações de um tipo. O primeiro registro fornece a sua capacidade de carga, o segundo a sua velocidade, o terceiro o seu custo fixo, e o quarto o seu custo variavel.
A saída do programa é o detalhamento de todas as viagens realizadas por todos os veículos para atender o problema, sendo especificado, para cada viagem, qual veículo da frota dimensionada a irá realizar, quais as suas características (capacidade, velocidade, custo fixo e variavel), informações da viagem (demanda total atendida na mesma, duração total da viagem, duração total de transbordo, a distância total percorrida, e o custo variavel da mesma, acrescido do custo fixo do veículo, se a viagem em questão for a primeira viagem do mesmo). Também é fornecido o custo total da solução obtida pelo programa.
Opcionalmente podem ser obtidos pelo programa:

- O número de nós fornecidos, o número de nós fictíceos criados, o número de veículos da frota, o número total de viagens realizadas, o custo total da solução, o tempo de processamento necessário para a obtenção da solução, a capacidade e a velocidade médias da frota.
- A composição da frota, detalhando para todos os veículos da mesma a capacidade, velocidade, custo fixo e variavel.
- O roteiro de todas as viagens de todos os veículos, especificando para cada nó as suas coordenadas X e Y, a sua demanda (ou duração máxima de jornada, se o mesmo for a base), a taxa de carregamento, se o mesmo é ou não a base, características do veículo que o atende (capacidade, velocidade de serviço, custos fixo e variavel), e se o nó em questão é o primeiro nó da primeira viagem do veículo. Essas informações são utilizadas pelo programa de melhoria de soluções.

Definições de registros

```
uses dos,crt;
type

{$IFDEF CPU87}
  {$N+}
  float = double;

{$ELSE}
  {$N-}
  float = real;

{$ENDIF}

  tipo_num  = array[0..100] of integer;

  arq_saida = record
                a : integer;
                b : integer;
                c : integer;
                d : integer;
                e : real;
                f : real;
                g : real;
                h : real;
              end;

  tipo_buf = record

               A : real;
               B : real;
               C : real;
               D : real;
             end;

  nos = record
          X   : float;
          Y   : float;
          Dem : float;
          Tc  : float;
        end;

  veiculos = record
               Cap : float;
               Vs  : float;
               Cf  : float;
               Co  : float;
             end;

  ptr_rel   = ^tipo_rel;
```

```
tipo_rel  = record
              tipo_vel   : integer;
              veiculo    : integer;
              demanda    : float;
              duracao    : float;
              dur_trans  : float;
              distancia  : float;
              custo      : float;
              f_redutor  : float;
              recomb     : boolean;
            end;

tipo_lista = array [1..100] of ptr_rel;

tipo_rota = record
              anterior   : integer;
              posterior  : integer;
              relatorio  : ptr_rel;
            end;

ptr_seq   = ^tipo_seq;

tipo_seq  = record
              final      : integer;
              relatorio  : ptr_rel;
              seguinte   : ptr_seq;
            end;

tipo_rede = record
              custo      : float;
              fechado    : boolean;
              anterior   : integer;
              arcos      : ptr_seq;
            end;

tipo_arquivo = string[12];
```

Definições de constantes do programa

```
const
  infinito      : float   = 9e+37;
  N_max         : integer = 100;
  M_max         : integer =  25;
  n_nos         : integer =   0;
  n_veiculos    : integer =   0;
  n_veic_frota  : integer =   0;
  Tv            : float   = 0.0;
  Fator_M       : integer =   0;
  ha_f_redutor  : boolean = false;
```

Variáveis globais do programa

```
var
  n_ficticeos : integer;
  no      : array[0..200] of nos;
  veiculo : array[1..25] of veiculos;
  dist    : array[1..5051] of float;
  rota    : array[0..100] of tipo_rota;
  rede    : array[0..100] of tipo_rede;
  n_viag  : integer;
  n_vel   : integer;
```

A função max determina o maior entre dois inteiros i e j

```
function max(i,j:integer):float;
begin
  if i>j then max:=i else max:=j;
end;
```

A função min determina o menor entre dois inteiros i e j

```
function min(i,j:integer):float;
begin
  if i>j then min:=j else min:=i;
end;
```

A função distância calcula a distância entre os nós i e j

```
function distancia(i,j:integer):float;
var
  h:integer;
begin
  i:=i+1;
  j:=j+1;
  if i>j then
  begin
    h:=i;
    i:=j;
    j:=h;
  end;
  h:=trunc(i+j*(j-1)/2);
  distancia:=dist[h];
end;
```

O procedimento armazena armazena valor na matriz de distâncias

```
procedure armazena(i,j:integer;valor:float);
var
  h:integer;
begin
```

```
    i:=i+1;
    j:=j+1;
    if i>j then
    begin
      h:=i;
      i:=j;
      j:=h;
    end;
    h:=trunc(i+j*(j-1)/2);
    dist[h]:=valor;
end;
```

O procedimento calcula_distancias cria a matriz de distâncias entre nós

```
procedure calcula_distancias;
var
  i,j : integer;
begin
  for i:=0 to n_nos do
    for j:=i+1 to n_nos do
      armazena(i,j,sqrt(sqr(no[i].X-no[j].X)+sqr(no[i].Y-no[j].Y)));
end;
```

O procedimento sort_veiculos ordena crescentemente o vetor veiculo segundo capacidade e velocidade, usando o procedimento insert sort.

```
procedure sort_veiculos;
var
  passe,local : integer;
  temp        : veiculos;
  achou       : boolean;
begin
  passe:=2;
  while passe<=n_veiculos do
  begin
    temp:=veiculo[passe];
    achou:=false;
    local:=passe-1;
    while (local>0) and not(achou) do
      if (veiculo[local].Cap>temp.Cap) or
         ((veiculo[local].Cap=temp.Cap) and (veiculo[local].Vs>temp.Vs))
then
      begin
        veiculo[local+1]:=veiculo[local];
        local:=local-1;
      end
      else
        achou:=true;
    veiculo[local+1]:=temp;
    passe:=passe+1;
  end;
```

```
    i:=i+1;
    j:=j+1;
    if i>j then
    begin
      h:=i;
      i:=j;
      j:=h;
    end;
    h:=trunc(i+j*(j-1)/2);
    dist[h]:=valor;
end;
```

O procedimento calcula_distancias cria a matriz de distâncias entre nós

```
procedure calcula_distancias;
var
  i,j : integer;
begin
  for i:=0 to n_nos do
    for j:=i+1 to n_nos do
      armazena(i,j,sqrt(sqr(no[i].X-no[j].X)+sqr(no[i].Y-no[j].Y)));
end;
```

O procedimento sort_veiculos ordena crescentemente o vetor veiculo segundo capacidade e velocidade, usando o procedimento insert sort.

```
procedure sort_veiculos;
var
  passe,local : integer;
  temp        : veiculos;
  achou       : boolean;
begin
  passe:=2;
  while passe<=n_veiculos do
  begin
    temp:=veiculo[passe];
    achou:=false;
    local:=passe-1;
    while (local>0) and not(achou) do
      if (veiculo[local].Cap>temp.Cap) or
         ((veiculo[local].Cap=temp.Cap) and (veiculo[local].Vs>temp.Vs))
then
      begin
        veiculo[local+1]:=veiculo[local];
        local:=local-1;
      end
      else
        achou:=true;
    veiculo[local+1]:=temp;
    passe:=passe+1;
  end;
```

```
end;
```

O procedimento le_arquivo lê um arquivo de nos, se e_no e verdadeiro, ou de veiculos, se e_no é falso

```
procedure le_arquivo(e_no:boolean;nome:tipo_arquivo;var sucesso:boolean)
var
  buffer  : tipo_buf;
  arquivo : file of tipo_buf;
  i       : integer;
begin
  sucesso:=true;
  if e_no then i:=0 else i:=1;
  assign(arquivo,nome);
{$I-}
  reset(arquivo);
{$I+}
  if IOresult<>0 then sucesso:=false;
  while not(eof(arquivo)) and sucesso do
  begin
    read(arquivo,buffer);
    if e_no then
    begin
      no[i].X:=buffer.A;
      no[i].Y:=buffer.B;
      no[i].Dem:=buffer.C;
      no[i].Tc:=buffer.D;
    end
    else
    begin
      veiculo[i].Cap:=buffer.A;
      veiculo[i].Vs:=buffer.B;
      veiculo[i].Cf:=buffer.C;
      veiculo[i].Co:=buffer.D;
    end;
    i:=i+1;
  end;
  if sucesso then
    if e_no then
    begin
      n_nos:=i-1;
      calcula_distancias;
      Tv:=no[0].Dem;
    end
    else
    begin
      n_veiculos:=i-1;
      sort_veiculos;
    end;
end;
```

O procedimento vizinho_mais_proximo cria um roteiro gigante inicial, utilizando a heurística de caixeiro viajante do vizinho mais próximo.

```
procedure Vizinho_mais_proximo;
var
  primeiro,
  proximo,
  i,j      : integer;
  minimo,
  dist     : float;
begin
  for i:=0 to n_nos do
  begin
    rota[i].anterior:=-1;
    rota[i].posterior:=-1;
    rota[i].relatorio:=nil;
  end;
  primeiro:=0;
  for i:=1 to n_nos do
  begin
    minimo:=infinito;
    proximo:=0;
    for j:=1 to n_nos do
    begin
      if (j<>primeiro) and (rota[j].anterior=-1) then
      begin
        dist:=distancia(primeiro,j);
        if dist<minimo then
        begin
          minimo:=dist;
          proximo:=j;
        end;
      end;
    end;
    rota[primeiro].posterior:=proximo;
    rota[proximo].anterior:=primeiro;
    primeiro:=proximo;
  end;
  rota[primeiro].posterior:=0;
  rota[0].anterior:=primeiro;
end;
```

O procedimento processe_2_opt realiza trocas sucessivas entre dois arcos de um roteiro fornecido, até que nenhuma nova troca seja possível, ou até que se tenha conseguido uma diminuição da distância total (sucesso é falso)

```
procedure processe_2_opt(var no_inicial:integer;var sucesso:boolean);
var
  comeco_arco_1,
  final_arco_1,
```

```
  comeco_arco_2,
  final_arco_2 : integer;
  i,j,k        : integer;
  reducao,
  acrescimo    : float;
  atual        : integer;
  temporario   : integer;
begin
  comeco_arco_1:=no_inicial;
  final_arco_1:=rota[comeco_arco_1].posterior;
  for i:=1 to n_nos-1 do
  begin
    comeco_arco_2:=rota[final_arco_1].posterior;
    final_arco_2:=rota[comeco_arco_2].posterior;
    for j:=i to n_nos-1 do
    begin
      reducao  :=distancia (comeco_arco_1,final_arco_1)+
                 distancia(comeco_arco_2,final_arco_2);
      acrescimo:=distancia(comeco_arco_1,comeco_arco_2)+
                 distancia(final_arco_1,final_arco_2);
      if reducao>acrescimo then
      begin
        rota[comeco_arco_1].posterior:=comeco_arco_2;
        rota[comeco_arco_2].posterior:=rota[comeco_arco_2].anterior;
        rota[comeco_arco_2].anterior:=comeco_arco_1;
        atual:=rota[comeco_arco_2].posterior;
        repeat
          temporario:=rota[atual].anterior;
          rota[atual].anterior:=rota[atual].posterior;
          rota[atual].posterior:=temporario;
          atual:=rota[atual].posterior;
        until atual=comeco_arco_1;
        rota[final_arco_1].posterior:=final_arco_2;
        rota[final_arco_2].anterior:=final_arco_1;
        sucesso:=false;
        no_inicial:=comeco_arco_1;
        exit;
      end;
      comeco_arco_2:=final_arco_2;
      final_arco_2:=rota[comeco_arco_2].posterior;
    end;
    comeco_arco_1:=final_arco_1;
    final_arco_1:=rota[comeco_arco_1].posterior;
  end;
  sucesso:=true;
end;
```

O procedimento dois_opt é a implementação computacional do algoritmo dois-ótimo de melhoria de roteiros de caixeiro viajante.

```
procedure dois_opt;
```

```
var
  no_inicial:integer;
  acabou     : boolean;
begin
  Vizinho_mais_proximo;
  acabou:=false;
  no_inicial:=0;
  while not acabou do
  begin
    processe_2_opt(no_inicial,acabou);
  end;
end;
```

O procedimento qual_veiculo obtém o tipo de veículo de menor custo para uma dada viagem, considerando ou não o fator redutor de custo fixo.

```
procedure qual_veiculo(redutor:boolean; uma_viagem:boolean;
                       duracao_transb:float; capacidade,velocidade,
                       distan : float; var melhor:integer;
                       var custo:float; var NV_final:integer;
                       var f_redutor_melhor:float);
var
  novo_custo : float;
  tipo       : integer;
  NV         : integer;
  f_redutor  : float;
begin
  melhor:=0;
  custo:=infinito;
  for tipo:=1 to n_veiculos do
  begin
    if uma_viagem then
    begin
      if veiculo[tipo].Cap<capacidade then
        NV:=0
      else
        NV:=1;
    end
    else
    begin
      if veiculo[tipo].Cap>=capacidade then
        NV:=1
      else
        if (trunc(capacidade) mod trunc(veiculo[tipo].Cap))=0 then
          NV:=trunc(capacidade/veiculo[tipo].Cap)-1
        else
          NV:=trunc(capacidade/veiculo[tipo].Cap);
    end;
    if  (NV>0)  and  (veiculo[tipo].Vs>=velocidade)  and  (velocidade>=0)
then
    begin
```

```
      if redutor then
      begin
        f_redutor:=duracao_transb+distan/veiculo[tipo].Vs;
        f_redutor:=1/trunc(Tv/f_redutor);
      end
      else
        f_redutor:=1;
      novo_custo:=NV*(f_redutor*veiculo[tipo].Cf+veiculo[tipo].Co*
                    distan);
      if novo_custo<custo then
      begin
        custo:=novo_custo;
        melhor:=tipo;
        NV_final:=NV;
        f_redutor_melhor:=f_redutor;
      end;
    end;
  end;
end;
```

O procedimento separa_nos_grandes decompoe nós com excesso de demanda em tantos nós fictícios quantos forem necessários, de forma que todos eles possam ser atendidos por pelo menos um tipo de veículo.

```
procedure separa_nos_grandes;
var
  f_redutor         : float;
  maior_capacidade  : float;
  NV                : integer;
  duracao_transb    : float;
  capacidade        : float;
  velocidade        : float;
  distan            : float;
  melhor            : integer;
  custo             : float;
  i,j               : integer;
begin
  n_ficticeos:=0;
  maior_capacidade:=veiculo[n_veiculos].Cap;
  for i:=1 to n_nos do
  begin
    if no[i].Dem>maior_capacidade then
    begin
      duracao_transb:=veiculo[n_veiculos].Cap/no[i].Tc+
                      veiculo[n_veiculos].Cap/no[0].Tc;
      distan:=2*distancia(0,i);
      velocidade:=distan/(Tv-duracao_transb);
      NV:=0;
      qual_veiculo(true,false,duracao_transb,no[i].Dem,
                   velocidade,distan,melhor,custo,NV,f_redutor);
      no[i].Dem:=no[i].Dem-NV*veiculo[melhor].Cap;
```

```
      for j:=1 to NV do
      begin
        n_ficticeos:=n_ficticeos+1;
        n_nos:=n_nos+1;
        no[n_nos].X:=no[i].X;
        no[n_nos].Y:=no[i].Y;
        no[n_nos].Dem:=veiculo[melhor].Cap;
        no[n_nos].Tc:=no[i].Tc;
      end;
      calcula_distancias;
    end;
  end;
end;
```

O procedimento determina_fator_M determina o número M de partições de roteiro gigante que serão realizadas.

```
procedure determina_Fator_M;
var
  transb,
  dist,
  fim_rota,
  veloc,
  custo,
  demanda  : float;
  NV,
  veic,
  i        : integer;
  continuo : boolean;
  redutor  : float;
begin
  Fator_M:=0;
  demanda:=0;
  transb:=0;
  i:=rota[0].posterior;
  dist:=distancia(0,i);
  continuo:=true;
  while continuo and (i<>0) do
  begin
    Fator_M:=Fator_M+1;
    fim_rota:=distancia(i,0);
    dist:=dist+fim_rota;
    demanda:=demanda+no[i].Dem;
    transb:=transb+no[i].Dem/no[i].Tc;
    veloc:=dist/(Tv-transb);
    qual_veiculo(false,true,transb,demanda,veloc,dist,veic,
                 custo,NV,redutor);
    if custo=infinito then
      continuo:=false;
    dist:=dist-fim_rota+distancia(i,rota[i].posterior);
    i:=rota[i].posterior;
```

```
  end;
  if Fator_M-1<1 then
    Fator_M:=1
  else
    Fator_M:=Fator_M-1;
end;
```

A função custo_da_viagem calcula o custo da viagem entre no_inicial e no_final realizada pelo veiculo de menor custo, utilizando ou não o fator redutor de custo fixo. Tambem determina a duração da viagem, a demanda atendida e qual o melhor tipo de veiculo, retornando essas informacoes como saída.

```
function custo_da_viagem(redutor:boolean;no_inicial,
                         no_final:integer):ptr_rel;

var
  no_atual,
  tipo           : integer;
  distan,
  capac,
  velocidade,
  custo,
  duracao_transb : float;
  relatorio      : ptr_rel;
  NV             : integer;
  f_redutor      : float;
begin
  no_atual:=no_inicial;
  distan:=0;
  capac:=0;
  duracao_transb:=0;
  if no_inicial<>no_final then
    while no_atual<>no_final do
    begin
      distan:=distan+distancia(no_atual,rota[no_atual].posterior);
      capac:=capac+no[no_atual].Dem;
      duracao_transb:=duracao_transb+no[no_atual].Dem/no[no_atual].Tc;
      no_atual:=rota[no_atual].posterior;
    end;
  capac:=capac+no[no_final].Dem;
  duracao_transb:=duracao_transb+no[no_final].Dem/no[no_final].Tc;
  duracao_transb:=duracao_transb+capac/no[0].Tc;
  distan:=distan+distancia(0,no_inicial)+distancia(0,no_final);
  if duracao_transb>=Tv then
  begin
    custo_da_viagem:=nil;
    exit;
  end;
  velocidade:=distan/(Tv-duracao_transb);
  qual_veiculo(redutor,true,duracao_transb,capac,velocidade,distan,
```

```
                tipo,custo,NV,f_redutor);
  if tipo=0 then
    custo_da_viagem:=nil
  else
  begin
    Getmem(relatorio,sizeof(tipo_rel));
    relatorio^.tipo_vel:=tipo;
    relatorio^.veiculo:=0;
    relatorio^.demanda:=capac;
    relatorio^.duracao:=distan/veiculo[tipo].Vs+duracao_transb;
    relatorio^.dur_trans:=duracao_transb;
    relatorio^.distancia:=distan;
    relatorio^.custo:=custo;
    relatorio^.f_redutor:=f_redutor;
    relatorio^.recomb:=true;
    custo_da_viagem:=relatorio;
  end;
end;
```

A função cria_rede cria uma rede de nos para calculo do caminho minimo entre os nos 0 e n_nos+1, devolvendo false se algum no não puder ser atendido nem por viagem exclusiva

```
function cria_rede(redutor:boolean):boolean;
var
  primeiro_no,
  segundo_no    : integer;
  buffer_custo : ptr_rel;
  arco_atual,
  buffer_seq    : ptr_seq;
  continuo      : boolean;
begin
  primeiro_no:=rota[0].posterior;
  rede[0].custo:=infinito;
  rede[0].anterior:=0;
  rede[0].fechado:=false;
  rede[0].arcos:=nil;
  while primeiro_no<>0 do
  begin
    rede[primeiro_no].custo:=infinito;
    rede[primeiro_no].anterior:=0;
    rede[primeiro_no].fechado:=false;
    rede[primeiro_no].arcos:=nil;
    segundo_no:=primeiro_no;
    continuo:=true;
    while (continuo) and (segundo_no<>0) do
    begin
      Getmem(buffer_seq,sizeof(tipo_seq));
      buffer_custo:=custo_da_viagem(redutor,primeiro_no,segundo_no);
      if (primeiro_no=segundo_no) and (buffer_custo=nil) then
      begin
```

```
        cria_rede:=false;
        exit;
      end;
      if buffer_custo=nil then
        continuo:=false
      else
      begin
        buffer_seq^.final:=rota[segundo_no].posterior;
        buffer_seq^.relatorio:=buffer_custo;
        buffer_seq^.seguinte:=nil;
        if primeiro_no=segundo_no then
          rede[primeiro_no].arcos:=buffer_seq
        else
          arco_atual^.seguinte:=buffer_seq;
        arco_atual:=buffer_seq;
      end;
      segundo_no:=rota[segundo_no].posterior;
    end;
    primeiro_no:=rota[primeiro_no].posterior;
  end;
  cria_rede:=true;
end;
```

A função dijkstra determina o caminho de menor custo entre o primeiro e o último nós da rede, devolvendo o custo desse caminho.

```
function Dijkstra:float;
var
  i,
  no_atual,
  no_rotulado : integer;
  ponteiro    : ptr_seq;
  custo_atual : float;
begin
  no_atual:=rota[0].posterior;
  rede[no_atual].custo:=0;
  custo_atual:=0;
  while no_atual<>0 do
  begin
    rede[no_atual].fechado:=true;
    ponteiro:=rede[no_atual].arcos;
    while ponteiro<>nil do
    begin
      no_rotulado:=ponteiro^.final;
      if (ponteiro^.relatorio^.custo+custo_atual)<
         rede[no_rotulado].custo then
      begin
        rede[no_rotulado].custo:=ponteiro^.relatorio^.custo+custo_atual;
        rede[no_rotulado].anterior:=no_atual;
      end;
      ponteiro:=ponteiro^.seguinte;
```

```
    end;
    custo_atual:=infinito;
    i:=0;
    repeat
      if not(rede[i].fechado) then
        if rede[i].custo<custo_atual then
        begin
          no_atual:=i;
          custo_atual:=rede[i].custo;
        end;
      i:=rota[i].posterior;
    until i=0;
    custo_atual:=rede[no_atual].custo;
  end;
  Dijkstra:=custo_atual;
end;
```

Realiza a partição do roteiro gigante em viagens

```
procedure Cria_Viagens;

var
  i,j       : integer;
  ponteiro :   ptr_seq;
begin
  i:=rota[0].posterior;
  repeat
    rota[i].relatorio:=nil;
    i:=rota[i].posterior;
  until i=0;
  i:=rede[0].anterior;
  j:=0;
  while i<>0 do
  begin
    ponteiro:=rede[i].arcos;
    while ponteiro^.final<>j do
      ponteiro:=ponteiro^.seguinte;
    rota[i].relatorio:=ponteiro^.relatorio;
    ponteiro^.relatorio:=nil;
    j:=i;
    i:=rede[i].anterior;
  end;
end;
```

O procedimento libera_rede é utilizado para minimizar o consumo de memória da C.P.U.

```
procedure libera_rede;
var
  ponteiro,
  temporario : ptr_seq;
```

```
  i          : integer;
begin
  i:=rota[0].posterior;
  while i<>0 do
  begin
    ponteiro:=rede[i].arcos;
    while (ponteiro<>nil) do
    begin
      if (ponteiro^.relatorio<>nil) then
        FreeMem(ponteiro^.relatorio,Sizeof(tipo_rel));
      temporario:=ponteiro;
      FreeMem(temporario,Sizeof(tipo_seq));
      ponteiro:=ponteiro^.seguinte;
    end;
    i:=rota[i].posterior;
  end;
end;
```

O procedimento sort_lista ordena a lista de viagens. É utilizado pelo procedimento combina_viagens.

```
procedure sort_lista(var lista:tipo_lista;inicio,fim: integer);
var
  passe,local : integer;
  temp        : ptr_rel;
  achou       : boolean;
begin
  passe:=2;
  while passe<=fim do
  begin
    temp:=lista[passe];
    achou:=false;
    local:=passe-1;
    while (local>0) and not(achou) do
      if (lista[local]^.demanda<temp^.demanda) or
         ((lista[local]^.demanda=temp^.demanda) and
         (veiculo[lista[local]^.tipo_vel].Vs<
         veiculo[temp^.tipo_vel].Vs)) then
      begin
        lista[local+1]:=lista[local];
        local:=local-1;
      end
      else
        achou:=true;
    lista[local+1]:=temp;
    passe:=passe+1;
  end;
end;
```

O procedimento combina_viagens procura diminuir o numero total de veículos na frota, alocando mais de uma viagem para cada veículo, quando possível

```
procedure combina_viagens;
var
  f_redutor,
  custo,
  velocidade,
  economia,
  distancia,
  dur_trans,
  maior_economia : float;
  lista          : tipo_lista;
  melhor,
  escolhido,
  NV,
  i,j,k,
  melhor_veiculo,
  atual,
  contador       : integer;
  combinou       : boolean;
  ponteiro       : ptr_rel;
  tecla          : char;
begin
  contador:=0;
  atual:=rota[0].posterior;
  while atual<>0 do
  begin
    if rota[atual].relatorio<>nil then
    begin
      contador:=contador+1;
      lista[contador]:=rota[atual].relatorio;
      lista[contador]^.veiculo:=0;
    end;
    atual:=rota[atual].posterior;
  end;
  writeln('QUICK SORT');
  sort_lista(lista,1,contador);
  writeln('COMBINANDO VIAGENS');
  for i:=1 to contador do
  begin
    if lista[i]^.veiculo=0 then
    begin
      repeat
        combinou:=false;
        maior_economia:=0;
        if contador=1 then
        begin
          lista[1]^.veiculo:=1;
        end;
```

```
for j:=i+1 to contador do
begin
  if lista[j]^.veiculo=0 then
  begin
    distancia:=0;
    dur_trans:=0;
    if lista[i]^.veiculo<>0 then
      for k:=i+1 to contador do
        if lista[k]^.veiculo=lista[i]^.veiculo then
        begin
          distancia:=distancia+lista[k]^.distancia;
          dur_trans:=dur_trans+lista[k]^.dur_trans;
        end;
    if (lista[i]^.dur_trans+lista[j]^.dur_trans+dur_trans>=
        Tv) then
      velocidade:=infinito
    else
      velocidade:=(lista[i]^.distancia+lista[j]^.distancia+
                   distancia)/(Tv-lista[i]^.dur_trans-lista[j]^.
                   dur_trans-dur_trans);
    qual_veiculo(false,true,0,lista[i]^.demanda,velocidade,
                 lista[i]^.distancia+
                 lista[j]^.distancia+distancia,
                 melhor,custo,NV,f_redutor);
    if custo<infinito then
    begin
      economia:=lista[i]^.custo+(1-lista[i]^.f_redutor)*
                veiculo[lista[i]^.tipo_vel].Cf+
                lista[j]^.custo+(1-lista[j]^.f_redutor)*
                veiculo[lista[j]^.tipo_vel].Cf-
                custo;
      if economia>maior_economia then
      begin
        escolhido:=j;
        maior_economia:=economia;
        melhor_veiculo:=melhor;
        combinou:=true;
      end;
    end;
  end;
end;
if combinou then
begin
  velocidade:=veiculo[melhor_veiculo].Vs;
  for j:=i to contador do
  begin
    if (j=i) and (lista[j]^.veiculo=0) then
      n_veic_frota:=n_veic_frota+1;
    if (j=i) or (j=escolhido) then
      lista[j]^.veiculo:=n_veic_frota;
    if (lista[j]^.veiculo=lista[i]^.veiculo) then
```

```
            begin
              lista[j]^.recomb:=false;
              lista[j]^.tipo_vel:=melhor_veiculo;
              lista[j]^.duracao:=lista[j]^.distancia/
                                 velocidade+lista[j]^.dur_trans;
              lista[j]^.f_redutor:=1;
              lista[j]^.custo:=veiculo[melhor_veiculo].Co*
                               lista[j]^.distancia;
              if j=i then
                lista[j]^.custo:=lista[j]^.custo+
                                 veiculo[melhor_veiculo].Cf;
            end;
          end;
        end;
      until combinou=false;
    end;
  end;
end;
```

A função cria_particao determina a melhor partição dos Fator_M roteiros fornecidos, retornando o seu custo.

```
function cria_particao(fator_redutor:boolean):float;
var
  custo_otimo,
  custo          : float;
  i,
  inicio,
  inicio_otimo,
  final,
  final_otimo    : integer;
  existe         : boolean;
begin
  custo_otimo:=infinito;
  inicio:=rota[0].posterior;
  final:=rota[0].anterior;
  for i:=1 to Fator_M do
  begin
    writeln(i,' : ',Fator_M);
    existe:=cria_rede(fator_redutor);
    if not(existe) then
    begin
      cria_particao:=infinito;
      exit;
    end;
    custo:=Dijkstra;
    if custo<custo_otimo then
    begin
      Cria_Viagens;
      custo_otimo:=custo;
      inicio_otimo:=inicio;
```

```
        final_otimo:=final;
      end;
      libera_rede;
      rota[0].posterior:=rota[inicio].posterior;
      rota[final].posterior:=inicio;
      rota[inicio].anterior:=final;
      rota[inicio].posterior:=0;
      rota[0].anterior:=inicio;
      inicio:=rota[0].posterior;
      final:=rota[0].anterior;
    end;
    rota[inicio].anterior:=final;
    rota[final].posterior:=inicio;
    rota[0].posterior:=inicio_otimo;
    rota[0].anterior:=final_otimo;
    rota[inicio_otimo].anterior:=0;
    rota[final_otimo].posterior:=0;
    if fator_redutor then
    begin
      writeln('COMBINA VIAGENS');
      combina_viagens;
      i:=rota[0].posterior;
      while i<>0 do
      begin
        if rota[i].relatorio^.f_redutor<1 then
          ha_f_redutor:=true;
        i:=rota[i].posterior;
      end;
    end;
    cria_particao:=custo_otimo;
end;
```

O procedimento refaz_particao refaz a partição de nós incluídos em viagens não combinadas com outras viagens

```
procedure refaz_particao;
var
  atual,
  inicio,
  fim,
  no_inicial,
  i,
  anterior,
  n_nos_real,
  primeiro,
  lixo,
  posterior   : integer;
  alterou,
  reconecta,
  acabou      : boolean;
  velocidade,
```

```
  custo      : float;
  ponteiro   : ptr_rel;
begin
  atual:=rota[0].posterior;
  primeiro:=atual;
  inicio:=atual;
  alterou:=false;
  while atual<>0 do
  begin
    if rota[atual].relatorio<>nil then
    begin
      if rota[atual].relatorio^.recomb then
      begin
        alterou:=true;
        if atual=inicio then
        begin
          repeat
            inicio:=rota[inicio].posterior;
          until (inicio=0) or (rota[inicio].relatorio<>nil);
          atual:=rota[inicio].anterior;
        end
        else
        begin
          anterior:=rota[atual].anterior;
          posterior:=rota[atual].posterior;
          while (posterior<>0) and (rota[posterior].relatorio=nil) do
            posterior:=rota[posterior].posterior;
          rota[rota[posterior].anterior].posterior:=rota[0].posterior;
          rota[rota[0].posterior].anterior:=rota[posterior].anterior;
          rota[0].posterior:=atual;
          rota[atual].anterior:=0;
          rota[anterior].posterior:=posterior;
          rota[posterior].anterior:=anterior;
          atual:=anterior;
        end;
      end;
    end;
    atual:=rota[atual].posterior;
  end;
  if (alterou) and (ha_f_redutor) then
  begin
    if primeiro=rota[0].posterior then
      reconecta:=false
    else
      reconecta:=true;
    if reconecta then
    begin
      writeln('FOI EMPREGADO O FATOR REDUTOR');
      fim:=rota[0].anterior;
      rota[0].anterior:=rota[inicio].anterior;
      rota[rota[inicio].anterior].posterior:=0;
```

```
    acabou:=false;
    no_inicial:=0;
    n_nos_real:=n_nos;
    n_nos:=0;
    i:=rota[0].posterior;
    repeat
      if rota[i].relatorio<>nil then
        Freemem(rota[i].relatorio,sizeof(tipo_rel));
      i:=rota[i].posterior;
      n_nos:=n_nos+1;
    until i=0;
    writeln('DOIS-OTIMO 2');
    while not acabou do
    begin
      processe_2_opt(no_inicial,acabou);
    end;
    determina_Fator_M;
    writeln('FATOR M = ',fator_M);
    writeln('CRIA PARTICAO');
    custo:=cria_particao(false);
    i:=rota[0].posterior;
    while i<>0 do
    begin
      if rota[i].relatorio<>nil then
      begin
        n_veic_frota:=n_veic_frota+1;
        rota[i].relatorio^.veiculo:=n_veic_frota;
      end;
      i:=rota[i].posterior;
    end;
  end
  else
  begin
    i:=rota[0].posterior;
    while i<>0 do
    begin
      if (rota[i].relatorio<>nil) and (rota[i].relatorio^.veiculo=0)
      then
      begin
        n_veic_frota:=n_veic_frota+1;
        ponteiro:=rota[i].relatorio;
        ponteiro^.veiculo:=n_veic_frota;
        velocidade:=ponteiro^.distancia/(Tv-ponteiro^.dur_trans);
        qual_veiculo(false,true,ponteiro^.dur_trans,ponteiro^.demanda,
                     velocidade,ponteiro^.distancia,ponteiro^.tipo_vel,
                     ponteiro^.custo,lixo,ponteiro^.f_redutor);
      end;
      i:=rota[i].posterior;
    end;
  end;
  if reconecta then
```

```
    begin
      anterior:=rota[0].anterior;
      rota[0].anterior:=fim;
      rota[fim].posterior:=0;
      rota[anterior].posterior:=inicio;
      rota[inicio].anterior:=anterior;
    end;
    n_nos:=n_nos_real;
  end;
end;
```

O procedimento escreve_relatorio grava em um arquivo em disco todas as viagens a serem realizadas pela frota.

```
procedure escreve_relatorio(arquivol:string);
type
  arq_relat = record
                X       : real;
                Y       : real;
                Dem     : real;
                Tc      : real;
                base    : boolean;
                inicio  : boolean;
                Cap     : real;
                Vs      : real;
                Cf      : real;
                Co      : real;
              end;
var
  relat     : file of arq_relat;
  bufrel    : arq_relat;
  alocado   : array [1..100] of boolean;
  i,j       : integer;
  quevel    : integer;
  primeiro  : boolean;
  ponteiro  : ptr_rel;
  OK        : boolean;
begin
  bufrel.X:=0;
  bufrel.Y:=0;
  bufrel.Dem:=0;
  bufrel.Tc:=0;
  bufrel.base:=false;
  bufrel.inicio:=false;
  bufrel.Cap:=0;
  bufrel.Vs:=0;
  bufrel.Cf:=0;
  bufrel.Co:=0;
  assign(relat,arquivol);
  rewrite(relat);
  for i:=1 to n_vel do
```

```
    alocado[i]:=false;
  i:=rota[0].posterior;
  while i<>0 do
  begin
    ponteiro:=rota[i].relatorio;
    if ponteiro<>nil then
    begin
      if not(alocado[ponteiro^.veiculo]) then
      begin
        quevel:=ponteiro^.veiculo;
        alocado[ponteiro^.veiculo]:=true;
        primeiro:=true;
        j:=i;
        while j<>0 do
        begin
          if rota[j].relatorio<>nil then
          begin
            if rota[j].relatorio^.veiculo=quevel then
            begin
              OK:=true;
              bufrel.Cap:=veiculo[ponteiro^.tipo_vel].Cap;
              bufrel.Vs:=veiculo[ponteiro^.tipo_vel].Vs;
              bufrel.Cf:=veiculo[ponteiro^.tipo_vel].Cf;
              bufrel.Co:=veiculo[ponteiro^.tipo_vel].Co;
              bufrel.base:=true;
              if primeiro then
              begin
                bufrel.inicio:=true;
                primeiro:=false;
              end
              else
              begin
                bufrel.inicio:=false;
              end;
              bufrel.X:=no[0].X;
              bufrel.Y:=no[0].Y;
              bufrel.Dem:=TV;
              bufrel.Tc:=no[0].Tc;
              write(relat,bufrel);
            end
            else
            begin
              OK:=false;
            end;
          end;
          if OK then
          begin
            bufrel.base:=false;
            bufrel.inicio:=false;
            bufrel.X:=no[j].X;
            bufrel.Y:=no[j].Y;
```

```
            bufrel.Dem:=no[j].Dem;
            bufrel.Tc:=no[j].Tc;
            write(relat,bufrel);
          end;
          j:=rota[j].posterior;
        end;
      end;
    end;
    i:=rota[i].posterior;
  end;
  close(relat);
end;
```

A rotina principal gerencia o programa de dimensionamento de frota (DIMEN.EXE) e apresenta a solução obtida.

```
var
  centesimos        : longint;
  i,j,k             : integer;
  custo             : float;
  ponteiro          : ptr_rel;
  custo_otimo       : float;
  saida             : file of arq_saida;
  buffer            : arq_saida;
  ss                : boolean;
  tecla             : char;
  erro,
  opcao             : integer;
  arquivo1,
  arquivo2          : string;
  velaloc           : array[1..100] of boolean;
  temp_proc         : real;
  h,m,s,c           : word;
  arquivo           : file of tipo_buf;
  buffer1           : tipo_buf;
  capacidade_media,
  velocidade_media  : real;
begin
  h:=0;
  m:=0;
  s:=0;
  c:=0;
  if paramcount<1 then
  begin
    writeln('DIMEN opcao [arquivo1] [arquivo2]');
    writeln('..... opcao = 1 -> saida na tela');
    writeln('      opcao = 2 -> saida resumida na tela');
    writeln('      opcao = 3 -> saida resumida em arquivo1');
    writeln('      opcao = 4 -> frota resultante em arquivo1');
    writeln('      opcao = 5 -> saida resumida em arquivo1');
    writeln('                   frota resultante em arquivo2');
```

```
    writeln('      opcao = 6 -> saida na tela e em arquivo1');
    exit;
  end;
  val(paramstr(1),opcao,erro);
  if (opcao>2) and (paramcount<2) then
  begin
    writeln('Falta <arquivo 1>');
    exit;
  end
  else
  begin
    arquivo1:=paramstr(2);
  end;
  if (opcao>4) and (opcao<6) and (paramcount<3) then
  begin
    writeln('Falta <arquivo 2>');
    exit;
  end
  else
  begin
    arquivo2:=paramstr(3);
  end;
  gettime(h,m,s,c);
  le_arquivo(true,'NOS.CMP',ss);
  le_arquivo(false,'VEICULOS.CMP',ss);
  writeln(n_nos);
  writeln('DECOMPOSICAO DE NOS COM EXCESSO DE DEMANDA');
  separa_nos_grandes;
  writeln('PROCEDIMENTO DOIS-OTIMO');
  dois_opt;
  i:=0;
  custo:=0;
  repeat
    custo:=custo+distancia(i,rota[i].posterior);
    i:=rota[i].posterior;
  until i=0;
  writeln('DETERMINACAO DO FATOR M');
  determina_Fator_M;
  writeln('CRIACAO DA PARTICAO COM FATOR REDUTOR');
  custo_otimo:=cria_particao(true);
  if custo_otimo=infinito then
  begin
    writeln('PROBLEMA SEM SOLUCAO');
    halt;
  end;
  writeln('REFAZ PARTICAO');
  refaz_particao;
  writeln('OTIMIZA VIAGENS');
  writeln('ACABOU');
  temp_proc:=3600*h+60*m+s+c/100;
  gettime(h,m,s,c);
```

```
temp_proc:=h*3600+60*m+s+c/100-temp_proc;
if (opcao>3) and (opcao<6) then
begin
  if opcao=4 then
    assign(arquivo,arquivo1)
  else
    assign(arquivo,arquivo2);
  rewrite(arquivo);
end;
for i:=1 to 100 do
  velaloc[i]:=false;
i:=rota[0].posterior;
n_viag:=0;
n_vel:=0;
n_nos:=0;
custo_otimo:=0;
while i<>0 do
begin
  ponteiro:=rota[i].relatorio;
  if ponteiro<>nil then
  begin
    if opcao=1 then
    begin
      writeln('Base');
      tecla:=readkey;
      clrscr;
    end;
    n_viag:=n_viag+1;
    if opcao=1 then
    begin
      writeln('VIAGEM ',n_viag);
      writeln('..FEITA PELO VEICULO NUMERO ',ponteiro^.veiculo);
    end;
    if not(velaloc[ponteiro^.veiculo]) then
    begin
      velaloc[ponteiro^.veiculo]:=true;
      n_vel:=n_vel+1;
      if (opcao>3) and (opcao<6) then
      begin
        buffer1.A:=veiculo[ponteiro^.tipo_vel].Cap;
        buffer1.B:=veiculo[ponteiro^.tipo_vel].Vs;
        buffer1.C:=veiculo[ponteiro^.tipo_vel].Cf;
        buffer1.D:=veiculo[ponteiro^.tipo_vel].Co;
        write(arquivo,buffer1);
      end;
    end;
    if opcao=1 then
    begin
      writeln('....Vs  = ',veiculo[ponteiro^.tipo_vel].Vs:10:2);
      writeln('....Cap = ',veiculo[ponteiro^.tipo_vel].Cap:10:2);
      writeln('....Cf  = ',veiculo[ponteiro^.tipo_vel].Cf:10:2);
```

```
        writeln('....Co  = ',veiculo[ponteiro^.tipo_vel].Co:10:2);
        writeln('..DEMANDA    = ',ponteiro^.demanda:10:2);
        writeln('..DURACAO    = ',ponteiro^.duracao:10:2);
        writeln('..TRANSBORDO = ',ponteiro^.dur_trans:10:2);
        writeln('..DISTANCIA  = ',ponteiro^.distancia:10:2);
        writeln('..CUSTO      = ',ponteiro^.custo:10:2);
        writeln('=============================================');
        write('Base - ');
      end;
      custo_otimo:=custo_otimo+ponteiro^.custo;
    end;
    if opcao=1 then
      write(i:3,' - ');
    if i<>0 then
      n_nos:=n_nos+1;
    i:=rota[i].posterior;
  end;
  if (opcao>3) and (opcao<6) then
    close(arquivo);
  if opcao=1 then
  begin
    writeln('Base');
    writeln('=============================================');
    writeln('CUSTO TOTAL : ',custo_otimo:10:2,' US$');
    writeln('=============================================');
  end;
  if (opcao=3) or (opcao=5) then
  begin
{$I-}
    assign(saida,arquivo1);
    reset(saida);
{$I+}
    if IOresult<>0 then
      rewrite(saida)
    else
      seek(saida,filesize(saida));
  end;
  capacidade_media:=0;
  velocidade_media:=0;
  for i:=1 to n_vel do
  begin
    for k:=1 to n_nos do
      if rota[k].relatorio^.veiculo=i then
        j:=k;
    capacidade_media:=capacidade_media+
                      veiculo[rota[j].relatorio^.tipo_vel].Cap;
    velocidade_media:=velocidade_media+
                      veiculo[rota[j].relatorio^.tipo_vel].Vs;
  end;
  capacidade_media:=capacidade_media/n_vel;
  velocidade_media:=velocidade_media/n_vel;
```

```
  if opcao=6 then escreve_relatorio(arquivol);
  if opcao>1 then
  begin
    writeln('Numero     Numero     Numero     Numero     Custo      Tempo');
    writeln('Nos        Ficticeos Veiculos  Viagens    Total      Proc.');
    writeln('-------------------------------------------------------------
             -');
    writeln((n_nos-n_ficticeos):10,n_ficticeos:10,n_vel:10,n_viag:10,
            custo_otimo:10:2,temp_proc:10:2);
    writeln('-------------------------------------------------------------
             -');
    writeln('CAPACIDADE MEDIA = ',capacidade_media:10:2);
    writeln('VELOCIDADE MEDIA = ',velocidade_media:10:2);
    if (opcao=3) or (opcao=5) then
    begin
      buffer.A:=n_nos-n_ficticeos;
      buffer.B:=n_ficticeos;
      buffer.C:=n_vel;
      buffer.D:=n_viag;
      buffer.E:=custo_otimo;
      {************************************************}
      if (temp_proc>0) and (temp_proc<100) then
        buffer.F:=temp_proc
      else
        buffer.F:=21;
      {************************************************}
      buffer.G:=capacidade_media;
      buffer.H:=velocidade_media;
      write(saida,buffer);
    end;
  end;
  if (opcao=3) or (opcao=5) then
    close(saida);
end.
```

II) PROGRAMA DE ALOCAÇÃO DE FROTA

O programa de alocação de frota é a implementação computacional do algoritmo de alocação de frota, exposto no capítulo 5.
Deve ser fornecido um arquivo, de quatro registros reais ,contendo todos os nós do problema a ser resolvido.
A primeira linha deve possuir informações da base da frota, o primeiro registro fornecendo a coordenada X, o segundo a coordenada Y, o quarto a taxa de carregamento da base. O terceiro registro fornece a duração máxima de jornada do problema.
As demais linhas devem possuir informações dos pontos de demanda da frota (uma linha para cada ponto de demanda). O primeiro registro fornece a coordenada X, o segundo a coordenada Y, o terceiro a sua demanda e o quarto a sua taxa de carregamento.
Deve também ser fornecido um arquivo, de quatro registros reais, contendo todos os veículos da frota que será utilizada para atender os pontos de demanda, cada linha contendo informações de um veículo. O primeiro registro fornece a sua capacidade de carga, o segundo a sua velocidade, o terceiro o seu custo fixo, e o quarto o seu custo variavel.
A saída do programa é o detalhamento de todas as viagens realizadas por cada um dos veículos da frota para atender o problema, sendo especificado, para cada veículo, as suas características (capacidade, velocidade, custo fixo e variavel), informações da sua jornada (demanda total atendida pelo mesmo, duração total de translado, duração total de transbordo, a duração total da jornada, e o custo total do veículo (fixo mais variável) para atender a todas as viagens que foram designadas ao mesmo). Também é fornecido o custo total da solução obtida pelo programa.
Opcionalmente podem ser obtidos pelo programa:

- O número de nós fornecidos, o número de nós fictíceos criados, o número de veículos da frota, o número total de viagens realizadas, o custo total da solução, o tempo de processamento necessário para a obtenção da solução, a capacidade e a velocidade médias da frota.
- O roteiro de todas as viagens de todos os veículos, especificando para cada nó as suas coordenadas X e Y, a sua demanda (ou duração máxima de jornada, se o mesmo for a base), a taxa de carregamento, se o mesmo é ou não a base, características do veículo que o atende (capacidade, velocidade de serviço, custos fixo e variavel), e se o nó em questão é o primeiro nó da primeira viagem do veículo. Essas informações são utilizadas pelo programa de melhoria de soluções.

Definições de registros

```
{$M $4000,0,$4000}
uses dos,crt;
type
{$IFDEF CPU87}
  {/$N+}
  float = double;
```

```
{$ELSE}
  {/$N-}
  float=real;
{$ENDIF}

  arq_saida = record
                a : integer;
                b : integer;
                c : integer;
                d : integer;
                e : real;
                f : real;
                g : real;
                h : real;
              end;

  tipo_2opt = record
                anterior  : integer;
                posterior : integer;
              end;

  tipo_arq  = string[255];

  tipo_buf  = record
                A : real;
                B : real;
                C : real;
                D : real;
              end;

  ptr_nos   = ^tipo_nos;

  tipo_nos  = record
                X       : float;
                Y       : float;
                Dem     : float;
                Tc      : float;
                alocado : boolean;
              end;

  ptr_rota  = ^tipo_rota;

  ptr_vel   = ^tipo_vel;

  tipo_rota = record
                no         : integer;
                relatorio  : ptr_vel;
                anterior   : ptr_rota;
                posterior  : ptr_rota;
              end;
```

```
  tipo_vel   = record
                 Cap          : float;
                 Vs           : float;
                 Cf           : float;
                 Co           : float;
                 unitario     : float;
                 demanda      : float;
                 viagem       : float;
                 transbordo   : float;
                 duracao      : float;
                 distancia    : float;
                 custo        : float;
                 alocado      : boolean;
                 roteiro      : ptr_rota;
               end;

arq_resumo   = record
                 X            : real;
                 Y            : real;
                 Dem          : real;
                 Tc           : real;
                 base         : boolean;
                 inicio       : boolean;
                 Cap          : real;
                 Vs           : real;
                 Cf           : real;
                 Co           : real;
               end;
```

Definição de constantes do programa

```
const
  infinito                  : float     = 9e+37;
  Tv                        : float     =   0.0;
  n_nos                     : integer   =     0;
  n_veic                    : integer   =     0;
  n_ficticeos               : integer   =     0;
  menor_cap                 : float     = 9e+37;
  maior_cap                 : float     =     0;
  adicionou_vel             : boolean   = false;
```

Definição de variáveis globais do programa

```
var
  h,m,s,c                   : word;
  duracao_otima,
  duracao_total             : float;
  processamento             : float;
  ultima_viagem             : ptr_rota;
  veiculo_retirado          : integer;
  relatorio_nulo            : ptr_vel;
```

```
  origem                    : ptr_rota;
  no                        : array [0..700] of ptr_nos;
  veiculo                   : array [1..700] of ptr_vel;
  dist                      : array[1..5051] of float;
  rota                      : array[0..700] of tipo_2opt;

  demanda_nao_atendida      : float;
  capacidade_disponivel     : float;
  deficit                   : float;
  opcao                     : integer;
  arquivol                  : string;
  capacidade_media,
  velocidade_media          : real;
  buffer_nao_atendidos      : arq_saida;
  conta_nao_aterndidos      : integer;
```

O procedimento mostra_viagens é responsável pela apresentação dos resultados obtidos pelo programa de alocação de frota.

```
procedure mostra_viagens;
var
  n_viagens: integer;
  ponteiro : ptr_rota;
  ctot     : float;
  tecla    : char;
  first    : boolean;
  buffer   : arq_saida;
  saida    : file of arq_saida;
  arqaux,
  resumo   : file of arq_resumo;
  buf_res  : arq_resumo;
begin
  first:=true;
  n_viagens:=0;
  n_veic:=0;
  capacidade_media:=0;
  velocidade_media:=0;
  ponteiro:=origem^.posterior;
  ctot:=0;
  if opcao=4 then
  begin
    assign(resumo,arquivol);
    rewrite(resumo);
  end;
  buf_res.X:=0;
  buf_res.Y:=0;
  buf_res.Dem:=0;
  buf_res.Tc:=0;
  buf_res.base:=false;
  buf_res.inicio:=false;
  buf_res.Cap:=0;
```

```
buf_res.Vs:=0;
buf_res.Cf:=0;
buf_res.Co:=0;
while ponteiro<>origem do
begin
  buf_res.base:=false;
  buf_res.inicio:=false;
  if ponteiro^.relatorio<>nil then
  begin
    if first then
      first:=false
    else
    begin
      if (opcao=1) or (opcao=4) then
      begin
        writeln(' : Base');
        if opcao=1 then
          tecla:=readkey;
      end;
    end;
    if (opcao=1) or (opcao=4) then
    begin
      clrscr;
      writeln('==========================================
               =========');
      writeln('Capacidade = ',ponteiro^.relatorio^.Cap:10:2);
      writeln('Vs         = ',ponteiro^.relatorio^.Vs:10:2);
      writeln('C. Fixo    = ',ponteiro^.relatorio^.Cf:10:2);
      writeln('C. Operac. = ',ponteiro^.relatorio^.Co:10:2);
      writeln('C. Unitar. = ',ponteiro^.relatorio^.unitario:10:2);
      writeln('------------------------------------------
               -----');
      writeln('Demanda    = ',ponteiro^.relatorio^.demanda:10:2);
      writeln('T. viagem  = ',ponteiro^.relatorio^.viagem:10:2);
      writeln('T. transb. = ',ponteiro^.relatorio^.transbordo:10:2);
      writeln('T. total   = ',ponteiro^.relatorio^.duracao:10:2);
      writeln('Custo      = ',ponteiro^.relatorio^.custo:10:2);
    end;
    if opcao=4 then
    begin
      buf_res.Cap:=ponteiro^.relatorio^.Cap;
      buf_res.Vs:=ponteiro^.relatorio^.Vs;
      buf_res.Cf:=ponteiro^.relatorio^.Cf;
      buf_res.Co:=ponteiro^.relatorio^.Co;
      buf_res.base:=true;
      buf_res.inicio:=true;
    end;
    capacidade_media:=capacidade_media+ponteiro^.relatorio^.Cap;
    velocidade_media:=velocidade_media+ponteiro^.relatorio^.Vs;
    ctot:=ctot+ponteiro^.relatorio^.custo+ponteiro^.relatorio^.Cf;
    n_viagens:=n_viagens+1;
```

```
      n_veic:=n_veic+1;
      if (opcao=1) or (opcao=4) then
      begin
        writeln('-------------------------------------------------------
                 ----');
        write('Base : ');
        buf_res.X:=no[0]^.X;
        buf_res.Y:=no[0]^.Y;
        buf_res.Dem:=TV;
        buf_res.Tc:=no[0]^.Tc;
      end;
    end
    else
    begin
      if ponteiro^.no=0 then
      begin
        n_viagens:=n_viagens+1;
        if (opcao=1) or (opcao=4) then
        begin
          writeln(' : Base');
          write('Base : ');
          buf_res.X:=no[0]^.X;
          buf_res.Y:=no[0]^.Y;
          buf_res.Dem:=TV;
          buf_res.Tc:=no[0]^.Tc;
          buf_res.base:=true;
          buf_res.inicio:=false;
        end;
      end
      else
      begin
        if (opcao=1) or (opcao=4) then
        begin
          write(ponteiro^.no:3);
          buf_res.X:=no[ponteiro^.no]^.X;
          buf_res.Y:=no[ponteiro^.no]^.Y;
          buf_res.Dem:=no[ponteiro^.no]^.Dem;
          buf_res.Tc:=no[ponteiro^.no]^.Tc;
          buf_res.base:=false;
          buf_res.inicio:=false;
        end;
      end;
    end;
    ponteiro:=ponteiro^.posterior;
    if opcao=4 then
      write(resumo,buf_res);
  end;
  if (opcao=4) and (adicionou_vel) then
  begin
    assign(arqaux,'PARTRES.CMP');
    reset(arqaux);
```

```
    while not(eof(arqaux)) do
    begin
      read(arqaux,buf_res);
      write(resumo,buf_res);
    end;
    close(arqaux);
  end;
  if opcao=4 then close(resumo);
  capacidade_media:=((capacidade_media)+(buffer_nao_atendidos.G*
                      buffer_nao_atendidos.C))/
                     (n_veic+buffer_nao_atendidos.C);
  velocidade_media:=((velocidade_media)+(buffer_nao_atendidos.H*
                      buffer_nao_atendidos.C))/
                     (n_veic+buffer_nao_atendidos.C);
  n_veic:=n_veic+buffer_nao_atendidos.C;
  n_viagens:=n_viagens+buffer_nao_atendidos.D;
  ctot:=ctot+buffer_nao_atendidos.E;
  if (opcao=1) then
  begin
    writeln(' : Base');
    writeln('=============================================================');
    writeln('CUSTO TOTAL : ',ctot:10:2);
    writeln('=============================================================');
    tecla:=readkey;
  end;
  if opcao>1 then
  begin
    writeln('Numero     Numero     Numero     Numero     Custo      Tempo');
    writeln('Nos        Ficticeos Veiculos  Viagens    Total      Proc.');
    writeln('-------------------------------------------------------------
             --');
    writeln((n_nos-n_ficticeos):10,n_ficticeos:10,n_veic:10,
             n_viagens:10,ctot:10:2,processamento:10:2);
    writeln('-------------------------------------------------------------
             --');
    writeln('CAPACIDADE MEDIA = ',capacidade_media:10:2);
    writeln('VELOCIDADE MEDIA = ',velocidade_media:10:2);
    if opcao=3 then
    begin
{$I-}
      assign(saida,arquivo1);
      reset(saida);
{$I+}
      if IOresult<>0 then
        rewrite(saida)
      else
        seek(saida,filesize(saida));
      buffer.A:=n_nos-n_ficticeos;
      buffer.B:=n_ficticeos;
      buffer.C:=n_veic;
      buffer.D:=n_viagens;
```

```
      buffer.E:=ctot;
      buffer.F:=processamento;
      buffer.G:=capacidade_media;
      buffer.H:=velocidade_media;
      write(saida,buffer);
      close(saida);
    end;
  end;
end;
```

A função distância calcula a distância entre os nós i e j

```
function distancia(i,j:integer):float;
var
  h : integer;
begin
  if i=j then
  begin
    distancia:=0;
    exit;
  end;
  i:=i+1;
  j:=j+1;
  if i>j then
  begin
    h:=i;
    i:=j;
    j:=h;
  end;
  h:=trunc(i+j*(j-1)/2);
  distancia:=dist[h];
end;
```

O procedimento armazena armazena valor na matriz de distâncias

```
procedure armazena(i,j:integer;valor:float);
var
  h : integer;
begin
  i:=i+1;
  j:=j+1;
  if i>j then
  begin
    h:=i;
    i:=j;
    j:=h;
  end;
  h:=trunc(i+j*(j-1)/2);
  dist[h]:=valor;
end;
```

O procedimento calcula_distancias cria a matriz de distâncias entre nós

```
procedure calcula_distancias;
var
  i,j : integer;
begin
  for i:=0 to n_nos do
    for j:=i+1 to n_nos do
      armazena(i,j,sqrt(sqr(no[i]^.X-no[j]^.X)+sqr(no[i]^.Y-no[j]^.Y)));
end;
```

O procedimento separa_nos_grandes decompõe nós com excesso de demanda em tantos nós fictícios quantos forem necessários, de forma que todos eles possam ser atendidos por pelo menos um tipo de veículo.

```
procedure separa_nos_grandes;
var
  original : integer;
  i,j      : integer;
  buf_no   : ptr_nos;
  NV       : integer;
begin
  if n_veic=0 then
  begin
    writeln('Carregar primeiro o arquivo de veiculos, depois o de nos');
    halt(0);
  end;
  n_ficticeos:=0;
  menor_cap:=infinito;
  maior_cap:=0;
  for i:=1 to n_veic do
  begin
    if veiculo[i]^.Cap<menor_cap then
      menor_cap:=veiculo[i]^.Cap;
    if veiculo[i]^.Cap>maior_cap then
      maior_cap:=veiculo[i]^.Cap;
  end;
  original:=n_nos;
  for i:=1 to original do
  begin
    if no[i]^.Dem>maior_cap then
    begin
      if ((no[i]^.Dem/menor_cap)-trunc(no[i]^.Dem/menor_cap))=0 then
        NV:=trunc(no[i]^.Dem/menor_cap)-1
      else
        NV:=trunc(no[i]^.Dem/menor_cap);
      no[i]^.Dem:=no[i]^.Dem-NV*menor_cap;
      for j:=1 to NV do
      begin
        n_ficticeos:=n_ficticeos+1;
        new(buf_no);
```

```
        buf_no^.X:=no[i]^.X;
        buf_no^.Y:=no[i]^.Y;
        buf_no^.Dem:=menor_cap;
        buf_no^.Tc:=no[i]^.Tc;
        buf_no^.alocado:=false;
        n_nos:=n_nos+1;
        no[n_nos]:=buf_no;
      end;
    end;
  end;
end;
```

O procedimento sort_veiculos ordena a lista de veículos por custos unitários crescentes, e por capacidades decrescentes, para um mesmo custo unitário.

```
procedure sort_veiculos;
var
  passe,local : integer;
  temp        : ptr_vei;
  achou       : boolean;
begin
  passe:=2;
  while passe<=n_veic do
  begin
    temp:=veiculo[passe];
    achou:=false;
    local:=passe-1;
    while (local>0) and not(achou) do
      if (veiculo[local]^.unitario>temp^.unitario) or
         ((veiculo[local]^.unitario=temp^.unitario) and
         (veiculo[local]^.Cap<temp^.Cap)) then
      begin
        veiculo[local+1]:=veiculo[local];
        local:=local-1;
      end
      else
        achou:=true;
    veiculo[local+1]:=temp;
    passe:=passe+1;
  end;
end;
```

O procedimento vizinho_mais_proximo cria um roteiro gigante inicial, utilizando a heurística de caixeiro viajante do vizinho mais próximo.

```
procedure Vizinho_mais_proximo;
var
  primeiro,
  proximo,
  i,j      : integer;
```

```
  minimo,
  dist      : float;
begin
  for i:=0 to n_nos do
  begin
    rota[i].anterior:=-1;
    rota[i].posterior:=-1;
  end;
  primeiro:=0;
  for i:=1 to n_nos do
  begin
    minimo:=infinito;
    proximo:=0;
    for j:=1 to n_nos do
    begin
      if (j<>primeiro) and (rota[j].anterior=-1) then
      begin
        dist:=distancia(primeiro,j);
        if dist<minimo then
        begin
          minimo:=dist;
          proximo:=j;
        end;
      end;
    end;
    rota[primeiro].posterior:=proximo;
    rota[proximo].anterior:=primeiro;
    primeiro:=proximo;
  end;
  rota[primeiro].posterior:=0;
  rota[0].anterior:=primeiro;
end;
```

O procedimento processe_2_opt realiza trocas sucessivas entre dois arcos de um roteiro fornecido, até que nenhuma nova troca seja possível, ou até que se tenha conseguido uma diminuição da distância total (sucesso é falso)

```
procedure processe_2_opt(var no_inicial:integer;var sucesso:boolean);
var
  comeco_arco_1,
  final_arco_1,
  comeco_arco_2,
  final_arco_2 : integer;
  i,j,k        : integer;
  reducao,
  acrescimo    : float;
  atual        : integer;
  temporario   : integer;
begin
  comeco_arco_1:=no_inicial;
```

```
  final_arco_1:=rota[comeco_arco_1].posterior;
  for i:=1 to n_nos-1 do
  begin
    comeco_arco_2:=rota[final_arco_1].posterior;
    final_arco_2:=rota[comeco_arco_2].posterior;
    for j:=i to n_nos-1 do
    begin
      reducao:=distancia(comeco_arco_1,final_arco_1)+
               distancia(comeco_arco_2,final_arco_2);
      acrescimo:=distancia(comeco_arco_1,comeco_arco_2)+
                 distancia(final_arco_1,final_arco_2);
      if reducao>acrescimo then
      begin
        rota[comeco_arco_1].posterior:=comeco_arco_2;
        rota[comeco_arco_2].posterior:=rota[comeco_arco_2].anterior;
        rota[comeco_arco_2].anterior:=comeco_arco_1;
        atual:=rota[comeco_arco_2].posterior;
        repeat
          temporario:=rota[atual].anterior;
          rota[atual].anterior:=rota[atual].posterior;
          rota[atual].posterior:=temporario;
          atual:=rota[atual].posterior;
        until atual=comeco_arco_1;
        rota[final_arco_1].posterior:=final_arco_2;
        rota[final_arco_2].anterior:=final_arco_1;
        sucesso:=false;
        no_inicial:=comeco_arco_1;
        exit;
      end;
      comeco_arco_2:=final_arco_2;
      final_arco_2:=rota[comeco_arco_2].posterior;
    end;
    comeco_arco_1:=final_arco_1;
    final_arco_1:=rota[comeco_arco_1].posterior;
  end;
  sucesso:=true;
end;
```

O procedimento ordena_nos cria uma lista de nós, utilizando para tanto os algoritmos de caixeiro viajante vizinho mais próximo e dois-ótimo.

```
procedure ordena_nos;
var
  no_inicial:integer;
  acabou    : boolean;
  auxiliar  : array [0..100] of ptr_nos;
  ponteiro  : integer;
  i         : integer;
begin
  Vizinho_mais_proximo;
  acabou:=false;
```

```
  no_inicial:=0;
  while not acabou do
  begin
    processe_2_opt(no_inicial,acabou);
  end;
  ponteiro:=rota[0].posterior;
  i:=1;
  while ponteiro<>0 do
  begin
    auxiliar[i]:=no[ponteiro];
    i:=i+1;
    ponteiro:=rota[ponteiro].posterior;
  end;
  for i:=1 to n_nos do
    no[i]:=auxiliar[i];
  calcula_distancias;
end;
```

O procedimento le_arquivo lê um arquivo de nos, se e_no e verdadeiro, ou de frota de veiculos, se e_no é falso

```
procedure le_arquivo(e_no:boolean;nome:tipo_arq;var sucesso:boolean);
var
  buffer  : tipo_buf;
  arquivo : file of tipo_buf;
  i       : integer;
  buf_no  : ptr_nos;
  buf_vel : ptr_vel;
begin
  if e_no then
    demanda_nao_atendida:=0
  else
    capacidade_disponivel:=0;
  sucesso:=true;
  if e_no then i:=0 else i:=1;
  assign(arquivo,nome);
{$I-}
  reset(arquivo);
{$I+}
  if IOresult<>0 then sucesso:=false;
  while not(eof(arquivo)) and sucesso do
  begin
    read(arquivo,buffer);
    if e_no then
    begin
      new(buf_no);
      buf_no^.X:=buffer.A;
      buf_no^.Y:=buffer.B;
      buf_no^.Dem:=buffer.C;
      if i<>0 then
        demanda_nao_atendida:=demanda_nao_atendida+buffer.C;
```

```
      buf_no^.Tc:=buffer.D;
      buf_no^.alocado:=false;
      no[i]:=buf_no;
    end
    else
    begin
      new(buf_vel);
      buf_vel^.Cap:=buffer.A;
      capacidade_disponivel:=capacidade_disponivel+buffer.A;
      buf_vel^.Vs:=buffer.B;
      buf_vel^.Cf:=buffer.C;
      buf_vel^.Co:=buffer.D;
      buf_vel^.unitario:=buffer.D/buffer.A;
      buf_vel^.demanda:=0;
      buf_vel^.viagem:=0;
      buf_vel^.transbordo:=0;
      buf_vel^.duracao:=0;
      buf_vel^.distancia:=0;
      buf_vel^.custo:=0;
      buf_vel^.alocado:=false;
      buf_vel^.roteiro:=nil;
      veiculo[i]:=buf_vel;
    end;
    i:=i+1;
  end;
  if sucesso then
    if e_no then
    begin
      n_nos:=i-1;
      Tv:=no[0]^.Dem;
      separa_nos_grandes;
      calcula_distancias;
      ordena_nos;
    end
    else
    begin
      n_veic:=i-1;
      sort_veiculos;
    end;
end;
```

O procedimento inicializa_algoritmo carrega os arquivos de nós e a frota de veículos, e inicializa parâmetros.

```
procedure inicializa_algoritmo;
var
  sucesso : boolean;
begin
  le_arquivo(false,'FROTA.CMP',sucesso);
  if not(sucesso) then halt(1);
  le_arquivo(true,'NOS.CMP',sucesso);
```

```
  if not(sucesso) then halt(1);
  ultima_viagem:=nil;
  veiculo_retirado:=0;
  deficit:=0;
  new(relatorio_nulo);
  relatorio_nulo^.cap:=0;
  relatorio_nulo^.Vs:=0;
  relatorio_nulo^.Cf:=0;
  relatorio_nulo^.Co:=0;
  relatorio_nulo^.unitario:=0;
  relatorio_nulo^.demanda:=0;
  relatorio_nulo^.viagem:=0;
  relatorio_nulo^.transbordo:=0;
  relatorio_nulo^.duracao:=0;
  relatorio_nulo^.distancia:=0;
  relatorio_nulo^.alocado:=false;
  new(origem);
  origem^.no:=0;
  origem^.relatorio:=relatorio_nulo;
  origem^.anterior:=origem;
  origem^.posterior:=origem;
  relatorio_nulo^.roteiro:=origem;
end;
```

A função algum_no_livre indica se existem ou não pontos de demanda não alocados a nenhum veículo.

```
function algum_no_livre:boolean;
var
  alocado : boolean;
  i       : integer;
begin
  alocado:=false;
  for i:=1 to n_nos do
    if (no[i]^.alocado=false) and (alocado=false) then
      alocado:=true;
  algum_no_livre:=alocado;
end;
```

A função algum_veículo_livre indica se existem ou não veículos com nenhum ponto de demanda alocado (ociosos).

```
function algum_veiculo_livre:boolean;
var
  alocado : boolean;
  i       : integer;
begin
  alocado:=false;
  for i:=1 to n_veic do
    if (veiculo[i]^.alocado=false) and (alocado=false) then
      alocado:=true;
```

```
  algum_veiculo_livre:=alocado;
end;
```

A função adiciona_nova_base inclue um retorno à base no roteiro de um veículo.

```
function adiciona_nova_base(posicao:ptr_rota;numero:integer):ptr_rota;
var
  base : ptr_rota;
begin
  new(base);
  base^.no:=0;
  base^.anterior:=posicao^.anterior;
  base^.posterior:=posicao;
  posicao^.anterior^.posterior:=base;
  posicao^.anterior:=base;
  if veiculo[numero]^.demanda=0 then
  begin
    base^.relatorio:=veiculo[numero];
    veiculo[numero]^.roteiro:=base;
  end
  else
  begin
    base^.relatorio:=nil;
  end;
  adiciona_nova_base:=base;
end;
```

A função verifica_todas_viagens é verdadeira se todas as viagens não violam nenhuma das restrições formuladas.

```
function verifica_todas_viagens(comeco:ptr_rota):boolean;
var
  ponteiro  : ptr_rota;
  duracao,
  demanda   : float;
  relatorio : ptr_vel;
begin
  ponteiro:=comeco;
  duracao_total:=0;
  duracao:=0;
  repeat
    if ponteiro^.relatorio<>nil then
    begin
      duracao_total:=duracao;
      duracao:=0;
      demanda:=0;
      relatorio:=ponteiro^.relatorio;
    end
    else
    begin
```

```
      if ponteiro^.no=0 then
      begin
        demanda:=0;
      end
      else
      begin
        duracao:=no[ponteiro^.no]^.Dem/no[ponteiro^.no]^.Tc+
                 no[ponteiro^.no]^.Dem/no[0]^.Tc+duracao;
        demanda:=demanda+no[ponteiro^.no]^.Dem;
      end;
    end;
    duracao:=duracao+distancia(ponteiro^.no,ponteiro^.posterior^.no)/
             relatorio^.Vs;
    if (duracao>Tv) or (demanda>relatorio^.Cap) then
    begin
      verifica_todas_viagens:=false;
      exit;
    end;
    ponteiro:=ponteiro^.posterior;
  until ponteiro=comeco;
  duracao_total:=duracao;
  verifica_todas_viagens:=true;
end;
```

A função aloca_nos procura alocar pontos de demanda na viagem cujo roteiro se inicia em inicio, sem violar as restrições de duração de jornada e de capacidade do veículo alocado à mesma. Não sendo possível alocar mais pontos de demanda em uma viagem, a função procura criar outras viagens para o veículo, através de retornos à base e alocação de pontos de demanda às novas viagens. A função aloca_nos é falsa se nenhum ponto de demanda foi alocado ao veículo que realiza a viagem que começa em inicio, verdadeiro em caso contrário.

```
function aloca_nos(inicio:ptr_rota;atual:integer):boolean;
var
  alocou_algum_no,
  incluiu_nova_base,
  alocou_outros_nos    : boolean;
  i                    : integer;
  novo_no,
  ponteiro,guarda      : ptr_rota;
  demanda,adicional,
  dur,
  distan,transb,otimo  : float;
  posicao              : ptr_rota;
  desconecta           : boolean;
label
  LOOP;
begin
  alocou_algum_no:=false;
  incluiu_nova_base:=false;
```

```
LOOP:
alocou_outros_nos:=false;
for i:=1 to n_nos do
begin
  if not(no[i]^.alocado) then
  begin
    ponteiro:=inicio;
    distan:=0;
    transb:=0;
    otimo:=infinito;
    repeat
      guarda:=ponteiro;
      demanda:=0;
      ponteiro:=ponteiro^.posterior;
      while ponteiro^.no<>0 do
      begin
        demanda:=demanda+no[ponteiro^.no]^.Dem;
        ponteiro:=ponteiro^.posterior;
      end;
      if (demanda+no[i]^.Dem<=veiculo[atual]^.Cap) then
      begin
        ponteiro:=guarda;
        repeat
          adicional:=distancia(ponteiro^.no,i)+
                     distancia(i,ponteiro^.posterior^.no)-
                     distancia(ponteiro^.no,ponteiro^.posterior^.no);
          dur:=veiculo[atual]^.duracao+adicional/veiculo[atual]^.Vs+
               no[i]^.Dem/no[i]^.Tc+no[i]^.Dem/no[0]^.Tc;
          if (adicional<otimo) and (dur<=Tv) then
          begin
            otimo:=adicional;
            posicao:=ponteiro;
          end;
          ponteiro:=ponteiro^.posterior;
        until ponteiro^.no=0;
      end;
    until ponteiro^.relatorio<>nil;
    if otimo<9e30 then
    begin
      alocou_algum_no:=true;
      alocou_outros_nos:=true;
      new(novo_no);
      novo_no^.no:=i;
      novo_no^.relatorio:=nil;
      novo_no^.anterior:=posicao;
      novo_no^.posterior:=posicao^.posterior;
      posicao^.posterior^.anterior:=novo_no;
      posicao^.posterior:=novo_no;
      no[i]^.alocado:=true;
      demanda_nao_atendida:=demanda_nao_atendida-no[i]^.Dem;
      veiculo[atual]^.demanda:=veiculo[atual]^.demanda+no[i]^.Dem;
```

```
      veiculo[atual]^.viagem:=veiculo[atual]^.viagem+
                              otimo/veiculo[atual]^.Vs;
      veiculo[atual]^.transbordo:=veiculo[atual]^.transbordo+
                                  no[i]^.Dem/no[i]^.Tc+
                                  no[i]^.Dem/no[0]^.Tc;
      veiculo[atual]^.duracao:=veiculo[atual]^.viagem+
                               veiculo[atual]^.transbordo;
      veiculo[atual]^.distancia:=veiculo[atual]^.distancia+otimo;
      veiculo[atual]^.custo:=veiculo[atual]^.custo+
                             veiculo[atual]^.Co*otimo;
    end;
  end;
end;
ponteiro:=inicio;
while ponteiro^.posterior^.relatorio=nil do
  ponteiro:=ponteiro^.posterior;
if not(alocou_outros_nos) and incluiu_nova_base then
begin
  ponteiro^.anterior^.posterior:=ponteiro^.posterior;
  ponteiro^.posterior^.anterior:=ponteiro^.anterior;
  dispose(ponteiro);
end
else
begin
  if inicio^.posterior^.relatorio=nil then
  begin
    ponteiro:=ponteiro^.posterior;
    novo_no:=adiciona_nova_base(ponteiro,atual);
    incluiu_nova_base:=true;
    goto LOOP;
  end;
end;
if origem^.posterior^.anterior=origem then
  desconecta:=true
else
  desconecta:=false;
if alocou_algum_no then
begin
  if desconecta then
  begin
    origem^.anterior^.posterior:=origem^.posterior;
    origem^.posterior^.anterior:=origem^.anterior;
  end;
  if not(verifica_todas_viagens(origem^.posterior)) then
  begin
    writeln('ERRO !!!');
    halt(0);
  end;
  if desconecta then
  begin
    origem^.anterior^.posterior:=origem;
```

```
      origem^.posterior^.anterior:=origem;
    end;
    duracao_otima:=duracao_total;
  end;
  aloca_nos:=alocou_algum_no;
end;
```

O procedimento desalocar_novo_veiculo tem por função minimizar o consumo de memória do programa, zerando parâmetros de veículos aos quais não pode ser alocado nenhum ponto de demanda.

```
procedure desalocar_novo_veiculo(atual:integer);
var
  destruidor : ptr_rota;
  alocou : boolean;
begin
  veiculo[atual]^.alocado:=false;
  destruidor:=veiculo[atual]^.roteiro;
  veiculo[atual]^.roteiro:=nil;
  destruidor^.anterior^.posterior:=destruidor^.posterior;
  destruidor^.posterior^.anterior:=destruidor^.anterior;
  dispose(destruidor);
end;
```

O procedimento alocar_novo_veiculo retira o veículo que ocupa a posição atual da lista e procura alocar ao mesmo tantos pontos de demanda quantos possíveis, criando para tanto quantas viagens forem necessárias.

```
procedure alocar_novo_veiculo(atual:integer);
var
  inicio : ptr_rota;
  alocou : boolean;
begin
  veiculo[atual]^.alocado:=true;
  inicio:=adiciona_nova_base(origem,atual);
  alocou:=aloca_nos(inicio,atual);
  if not(alocou) then desalocar_novo_veiculo(atual);
  capacidade_disponivel:=capacidade_disponivel-veiculo[atual]^.Cap;
  deficit:=demanda_nao_atendida-capacidade_disponivel;
end;
```

A função testademanda é verdadeira se o ponto de maior demanda não alocado a nenhuma viagem pode substituir o ponto de demanda indicado por ponteiro sem violar a capacidade do veículo.

```
function
testademanda(ponteiro:ptr_rota;maior:float;relatorio:ptr_vel):boolean;
var
  auxiliar : ptr_rota;
  demanda  : float;
```

```
begin
  auxiliar:=ponteiro;
  while auxiliar^.anterior^.no<>0 do
    auxiliar:=auxiliar^.anterior;
  demanda:=0;
  while auxiliar^.no<>0 do
  begin
    if auxiliar=ponteiro then
      demanda:=demanda+maior
    else
      demanda:=demanda+no[auxiliar^.no]^.Dem;
    auxiliar:=auxiliar^.posterior;
  end;
  if demanda>relatorio^.Cap then
    testademanda:=false
  else
    testademanda:=true;
end;
```

O procedimento permuta_nos procura diminuir o deficit de demanda de um problema, procurando alocar pontos de maior demanda em substituição a outros pontos de demanda das viagens já criadas pelo algoritmo, e também procurando alocar mais pontos de demanda às viagens sempre que houver alguma substituição.

```
procedure permuta_nos;
var
  conseguiu   : boolean;
  maior       : float;
  i,
  qual        : integer;
  pos_troca,
  ponteiro    : ptr_rota;
  demanda     : float;
  pos_rel,
  relatorio   : ptr_vel;
  addur,
  adtrans,
  reddur,
  redtrans    : float;
  distan,
  transbordo  : float;
  vel         : integer;
begin
  repeat
    conseguiu:=false;
    maior:=0;
    qual:=0;
    for i:=1 to n_nos do
    begin
      if not(no[i]^.alocado) then
```

```
    begin
      if no[i]^.Dem>maior then
      begin
        maior:=no[i]^.Dem;
        qual:=i;
      end;
    end;
  end;
  ponteiro:=origem^.posterior;
  demanda:=9e31;
  while ponteiro<>origem do
  begin
    if ponteiro^.relatorio<>nil then
      relatorio:=ponteiro^.relatorio;
    if (ponteiro^.no<>0) and (no[ponteiro^.no]^.Dem<maior) and
       (testademanda(ponteiro,maior,relatorio)) then
    begin
      addur:=distancia(ponteiro^.anterior^.no,qual)+
             distancia(qual,ponteiro^.posterior^.no);
      adtrans:=no[qual]^.Dem/no[qual]^.Tc+
               no[qual]^.Dem/no[0]^.Tc;
      reddur:=distancia(ponteiro^.anterior^.no,ponteiro^.no)+
              distancia(ponteiro^.no,ponteiro^.posterior^.no);
      redtrans:=no[ponteiro^.no]^.Dem/no[ponteiro^.no]^.Tc+
                no[ponteiro^.no]^.Dem/no[0]^.Tc;
      if (relatorio^.duracao+(addur-reddur)/relatorio^.Vs+
          adtrans-redtrans)<=Tv then
      begin
        if (no[ponteiro^.no]^.Dem<demanda) then
        begin
          demanda:=no[ponteiro^.no]^.Dem;
          pos_rel:=relatorio;
          pos_troca:=ponteiro;
          distan:=addur-reddur;
          transbordo:=adtrans-redtrans;
          conseguiu:=true;
        end;
      end;
    end;
    ponteiro:=ponteiro^.posterior;
  end;
  if conseguiu then
  begin
    no[qual]^.alocado:=true;
    no[pos_troca^.no]^.alocado:=false;
    demanda_nao_atendida:=demanda_nao_atendida-no[qual]^.Dem+
                          no[pos_troca^.no]^.Dem;
    deficit:=demanda_nao_atendida-capacidade_disponivel;
    pos_rel^.demanda:=pos_rel^.demanda+no[qual]^.Dem-
                      no[pos_troca^.no]^.Dem;
    pos_troca^.no:=qual;              {Troca os nos}
```

```
      pos_rel^.viagem:=pos_rel^.viagem+distan/pos_rel^.Vs;
      pos_rel^.transbordo:=pos_rel^.transbordo+transbordo;
      pos_rel^.duracao:=pos_rel^.viagem+pos_rel^.transbordo;
      pos_rel^.distancia:=pos_rel^.distancia+distan;
      pos_rel^.custo:=pos_rel^.custo+distan*pos_rel^.Co;
    end;
    ponteiro:=origem^.posterior;
    while ponteiro<>origem do
    begin
      if ponteiro^.relatorio<>nil then
      begin
        vel:=1;
        while veiculo[vel]<>ponteiro^.relatorio do
          vel:=vel+1;
        if aloca_nos(ponteiro,vel) then
        begin
          writeln('CONSEGUIU ALOCAR MAIS NOS');
          conseguiu:=true;
        end;
      end;
      ponteiro:=ponteiro^.posterior;
    end;
    deficit:=demanda_nao_atendida-capacidade_disponivel;
  until (not(conseguiu)) or (deficit=0);
end;
```

A função loop_principal é a implementação do algoritmo de alocação de frota, sendo verdadeira se todos os pontos de demanda foram alocados a viagens, e falsa em caso contrário.

```
function loop_principal:boolean;
var
  conseguiu : boolean;
  tecla     : char;
begin
  writeln('INICIALIZANDO');
  inicializa_algoritmo;
  veiculo_retirado:=0;
  deficit:=demanda_nao_atendida-capacidade_disponivel;
  while algum_no_livre and algum_veiculo_livre do
  begin
    veiculo_retirado:=veiculo_retirado+1;
    writeln('ALOCANDO NOVO VEICULO');
    alocar_novo_veiculo(veiculo_retirado);
    permuta_nos;
    deficit:=demanda_nao_atendida-capacidade_disponivel;
  end;
  if deficit>0 then
    permuta_nos;
  if deficit>0 then
  begin
```

```
    writeln('A FROTA E INSUFICIENTE');
    conseguiu:=false;
  end
  else
  begin
    writeln('ALOCACAO PERFEITA');
    conseguiu:=true;
  end;
  loop_principal:=conseguiu;
end;
```

A rotina principal executa o algoritmo de alocação de frota, apresenta a solução, e adiciona veículos à frota, através do programa de dimensionamento de frota, se existiren pontos de demanda órfãos, alocando os novos veículos aos mesmos.

```
var
  i     : integer;
  tecla : char;
  erro  : integer;
  b     : tipo_buf;
  a     : file of tipo_buf;
  a_ad  : file of arq_saida;
begin
  adicionou_vel:=false;
  if paramcount<1 then
  begin
    writeln('ALOCA opcao [arquivo1]');
    writeln('..... opcao = 1 -> saida na tela');
    writeln('       opcao = 2 -> saida resumida na tela');
    writeln('       opcao = 3 -> saida resumida em arquivo1');
    writeln('       opcao = 4 -> saida na tela, completa em arquivo 1');
    exit;
  end;
  val(paramstr(1),opcao,erro);
  if (opcao>2) and (paramcount<2) then
  begin
    writeln('Falta [arquivo 1]');
    exit;
  end
  else
  begin
    arquivo1:=paramstr(2);
  end;
  buffer_nao_atendidos.A:=0;
  buffer_nao_atendidos.B:=0;
  buffer_nao_atendidos.C:=0;
  buffer_nao_atendidos.D:=0;
  buffer_nao_atendidos.E:=0;
  buffer_nao_atendidos.F:=0;
  buffer_nao_atendidos.G:=0;
```

```
buffer_nao_atendidos.H:=0;
gettime(h,m,s,c);
if not(loop_principal) then
begin
  writeln('Os seguintes nos nao foram alocados:');
  assign(a,'NOS.CMP');
  rewrite(a);
  b.A:=no[0]^.X;
  b.B:=no[0]^.Y;
  b.C:=no[0]^.Dem;
  b.D:=no[0]^.Tc;
  write(a,b);
  for i:=1 to n_nos do
  begin
    if not(no[i]^.alocado) then
    begin
      writeln('-> ',i);
      b.A:=no[i]^.X;
      b.B:=no[i]^.Y;
      b.C:=no[i]^.Dem;
      b.D:=no[i]^.Tc;
      write(a,b);
    end;
  end;
  close(a);
  writeln('RODANDO DIMEN.EXE .....');
  exec('DIMEN.EXE',' 6 PARTRES.CMP');
  assign(a_ad,'ADICIONA.CMP');
  rewrite(a_ad);
  close(a_ad);
  exec('DIMEN.EXE',' 3 ADICIONA.CMP');
  if doserror<>0 then
  begin
    write(doserror);
    halt(0);
  end;
  if doserror<>0 then
  begin
    write(doserror);
    halt(0);
  end;
  assign(a_ad,'ADICIONA.CMP');
  adicionou_vel:=true;
  reset(a_ad);
  read(a_ad,buffer_nao_atendidos);
  close(a_ad);
end;
processamento:=h*3600+m*60+s+c/100;
gettime(h,m,s,c);
processamento:=h*3600+m*60+s+c/100-processamento;
if opcao=1 then
```

```
    tecla:=readkey;
  mostra_viagens;
end.
```

III) PROGRAMA DE MELHORIA DE SOLUÇÕES

O programa de melhoria de soluções é a implementação computacional do algoritmo de mesmo nome, proposto no capítulo 6.
A entrada do programa é um arquivo contendo todas as viagens realizadas por todos os veículos de uma frota utilizada para atender a um conjunto de pontos de demanda. Cada linha do arquivo corresponde a um nó (ponto de demanda ou base). O primeiro registro de cada linha contém a coordenada X do nó, o segundo registro a sua coordenada Y. O terceiro registro contém a demanda do nó (se for um ponto de demanda) ou a duração máxima de jornada (se for a base). O quarto registro contém a taxa de carregamento do nó. O quinto e o sexto registros são variáveis booleanas: o quinto é verdadeiro se o nó correspodente for a base, o sexto é verdadeiro se o nó correspodente for o primeiro nó atendido na primeira viagem realizada por um veículo (ou seja, o veículo em questão não atendeu a nenhum dos nós anteriores do arquivo). Os últimos quatro registros contém dados do veículo que atende o nó: o sétimo contéma sua capacidade, o oitavo a sua velocidade de serviço, o nono o custo fixo e o décimo o custo variável.
O algoritmo apresenta como saída, para todos os veículos não ociosos da frota, as características do mesmo (capacidade, velocidade, custos fixo e variável), e, para cada viagem do veículo, a demanda atendida na mesma, a sua duração total, o seu custo total, e também o custo acumulado de todas as viagens anteriores de todos os veículos já apresentados, somados ao custo total da mesma (custo acumulado da solução).

Definições de registros

```
uses dos,crt;

type

  tipo_sentido = (frente,tras);

  tipo_buffer = record
                  X      : real;
                  Y      : real;
                  Dem    : real;
                  Tc     : real;
                  base   : boolean;
                  inicio : boolean;
                  Cap    : real;
                  Vs     : real;
                  Cf     : real;
                  Co     : real;
                end;

  ptr_buffer  = ^tipo_buffer;
```

```
  ptr_rede    = ^tipo_rede;

  tipo_rede   = record
                  anterior  : ptr_rede;
                  posterior : ptr_rede;
                  buffer    : ptr_buffer;
                end;
```

Definição de variáveis globais

```
var
  guia        : ptr_rede;
  Tv          : real;
  Tcbase      : real;
```

O procedimento le_arquivo carrega a solução inicial

```
procedure le_arquivo;
var
  arquivo  : file of tipo_buffer;
  buffer   : tipo_buffer;
  ponteiro,
  temp1    : ptr_rede;
  temp2    : ptr_buffer;
  primeiro : boolean;
begin
  assign(arquivo,'RESULT.CMP');
  reset(arquivo);
  primeiro:=true;
  ponteiro:=nil;
  guia:=nil;
  Tv:=0;
  Tcbase:=0;
  while not(eof(arquivo)) do
  begin
    read(arquivo,buffer);
    new(temp1);
    new(temp2);
    temp2^.X:=buffer.X;
    temp2^.Y:=buffer.Y;
    temp2^.Dem:=buffer.Dem;
    temp2^.Tc:=buffer.Tc;
    temp2^.base:=buffer.base;
    if (temp2^.base) and (Tv=0) then
      Tv:=temp2^.Dem;
    if (temp2^.base) and (Tcbase=0) then
      Tcbase:=temp2^.Tc;
    temp2^.inicio:=buffer.inicio;
    temp2^.Cap:=buffer.Cap;
    temp2^.Vs:=buffer.Vs;
```

```
    temp2^.Cf:=buffer.Cf;
    temp2^.Co:=buffer.Co;
    if primeiro then
    begin
      primeiro:=false;
      guia:=temp1;
      temp1^.anterior:=nil;
      temp1^.posterior:=nil;
      temp1^.buffer:=temp2;
      ponteiro:=temp1;
    end
    else
    begin
      temp1^.anterior:=ponteiro;
      temp1^.posterior:=nil;
      temp1^.anterior^.posterior:=temp1;
      temp1^.buffer:=temp2;
      ponteiro:=temp1;
    end;
  end;
  guia^.anterior:=ponteiro;
  ponteiro^.posterior:=guia;
  close(arquivo);
end;
```

A função distância obtém a distância entre os nós 1 e 2

```
function distancia(no1,no2:ptr_rede):real;
begin
  distancia:=sqrt(sqr(no1^.buffer^.X-no2^.buffer^.X)+
                  sqr(no1^.buffer^.Y-no2^.buffer^.Y));
end;
```

O procedimento testa_rede percorre o roteiro gigante de viagens do nó comeco_2 ao nó final_1, verificando se as viagens contidas nesse trecho não violam as restrições de capacidade dos veículos que as realizam e de duração máxima de jornada

```
procedure testa_rede(final_1,comeco_2:ptr_rede;var custo:real;
                     var erro:boolean);
var
  demanda,
  duracao      : real;
  ponteiro     : ptr_rede;
  Cap,Vs,Co    : real;
  buffer       : real;
begin
  demanda:=0;
  duracao:=0;
  custo:=0;
  erro:=false;
```

```
  ponteiro:=guia;
  repeat
    if ponteiro^.buffer^.base then
    begin
      demanda:=0;
      if ponteiro^.buffer^.inicio then
      begin
        duracao:=0;
        Cap:=ponteiro^.buffer^.Cap;
        Vs:=ponteiro^.buffer^.Vs;
        Co:=ponteiro^.buffer^.Co;
        if not(ponteiro^.posterior^.buffer^.inicio) then
          custo:=custo+ponteiro^.buffer^.Cf;
      end;
    end;
    buffer:=distancia(ponteiro,ponteiro^.posterior);
    if not(ponteiro^.buffer^.base) then
    begin
      demanda:=demanda+ponteiro^.buffer^.Dem;
      duracao:=duracao+
               buffer/Vs+
               ponteiro^.buffer^.Dem/ponteiro^.buffer^.Tc+
               ponteiro^.buffer^.Dem/Tcbase;
      custo:=custo+buffer*Co;
    end;
    if (ponteiro^.buffer^.base) then
    begin
      duracao:=duracao+buffer/Vs;
      custo:=custo+buffer*Co;
    end;
    if (demanda>Cap) or (duracao>Tv) then
    begin
      custo:=9e+37;
      erro:=true;
      exit;
    end;
    ponteiro:=ponteiro^.posterior;
  until ponteiro=guia;
  erro:=false;
end;
```

O procedimento troca_arcos faz parte do procedimento dois_otimo, e troca os arcos que conectam os nós comeco_1 a final_1 e comeco_2 a final_2

```
procedure troca_arcos(final_1,comeco_2:ptr_rede);
var
  comeco,
  final,
  ponteiro,
  buffer    : ptr_rede;
begin
```

```
  comeco:=final_1^.anterior;
  final:=comeco_2^.posterior;
  ponteiro:=comeco_2;
  repeat
    buffer:=ponteiro^.anterior;
    ponteiro^.anterior:=ponteiro^.posterior;
    ponteiro^.posterior:=buffer;
    ponteiro:=ponteiro^.posterior;
  until ponteiro=comeco;
  comeco^.posterior:=comeco_2;
  comeco_2^.anterior:=comeco;
  final^.anterior:=final_1;
  final_1^.posterior:=final;
end;
```

A função reduz_custo é verdadeira se a troca dos arcos que conectam os nós comeco_1 a final_1 e comeco_2 a final_2 não viola nenhuma restrição do problema (capacidade e duração de jornada)

```
function reduz_custo(final_1,comeco_2:ptr_rede):boolean;
var
  custo_inicial,
  novo_custo     : real;
  erro           : boolean;
  sucesso        : boolean;
begin
  testa_rede(final_1,comeco_2,custo_inicial,erro);
  if erro then
  begin
    writeln('ERRO NO PROCESSO DE TROCA!!!!!');
    halt(0);
  end;
  troca_arcos(final_1,comeco_2);
  testa_rede(comeco_2,final_1,novo_custo,erro); {Cuidado, Inverteu!!}
  if (erro=true) or (trunc(novo_custo*100)>=trunc(custo_inicial*100))
  then
  begin
    troca_arcos(comeco_2,final_1);
    sucesso:=false;
  end
  else
  begin
    sucesso:=true;
    writeln('Melhorou solucao');
  end;
  reduz_custo:=sucesso;
end;
```

O procedimento dois_opt é a implementação computacional do algoritmo dois-ótimo para redução do custo operacional do roteiro gigante de

viagens, devolvendo sucesso verdadeiro se foi possivel reduzir o custo operacional com uma troca de arcos.

```
procedure dois_opt(var no_inicial:ptr_rede;var sucesso:boolean);
var
  comeco_1,final_1,
  comeco_2,final_2 : ptr_rede;
begin
  writeln('Executando otimizacao');
  comeco_1:=no_inicial;
  final_1:=comeco_1^.posterior;
  sucesso:=false;
  repeat
    writeln('Proximo par');
    comeco_2:=final_1^.posterior;
    final_2:=comeco_2^.posterior;
    repeat
      write('.');
      sucesso:=reduz_custo(final_1,comeco_2);
      if not(sucesso) then
      begin
        comeco_2:=comeco_2^.posterior;
        final_2:=final_2^.posterior;
      end;
    until (final_2=no_inicial) or (sucesso=true);
    writeln('');
    if not(sucesso) then
    begin
      comeco_1:=comeco_1^.posterior;
      final_1:=comeco_1^.posterior;
    end;
  until (final_1^.posterior^.posterior=no_inicial) or (sucesso=true);
end;
```

O procedimento reagrupar executa o procedimento dois_opt até que nenhuma redução adicional do custo operacional seja possível pela troca simultânea de dois arcos

```
procedure reagrupar;
var
  melhorou    : boolean;
  no_inicial : ptr_rede;
begin
  melhorou:=true;
  no_inicial:=guia;
  while melhorou do dois_opt(no_inicial,melhorou);
end;
```

O procedimento elimina_veiculos_inuteis elimina da frota veículos ociosos

```
procedure elimina_veiculos_inuteis;
var
  ponteiro : ptr_rede;
  tecla    : char;
begin
  ponteiro:=guia;
  repeat
    if ponteiro^.buffer^.inicio and
       ponteiro^.posterior^.buffer^.inicio then
    begin
      if ponteiro=guia then
        guia:=ponteiro^.posterior;
      writeln('REDUZIU FROTA');
      ponteiro:=ponteiro^.posterior;
      ponteiro^.anterior^.anterior^.posterior:=ponteiro;
      ponteiro^.anterior:=ponteiro^.anterior^.anterior;
    end
    else
    begin
      ponteiro:=ponteiro^.posterior;
    end;
  until ponteiro=guia;
end;
```

O procedimento mostrar_solucao apresenta a solução melhorada na tela do computador

```
procedure mostrar_solucao;
var
  descricao,
  ponteiro     : ptr_rede;
  custo_total,
  custo_viagem,
  demanda,
  duracao      : real;
  n_vel        : integer;
  primeiro     : boolean;
  tecla        : char;
  buffer       : real;
begin
  ponteiro:=guia;
  custo_total:=0;
  custo_viagem:=0;
  demanda:=0;
  duracao:=0;
  n_vel:=0;
  primeiro:=true;
  repeat
    if ponteiro^.buffer^.inicio then
    begin
```

```
      n_vel:=n_vel+1;
      if not(primeiro) then
      begin
        writeln('Demanda atendida : ',demanda:10:2);
        writeln('Duracao total    : ',duracao:10:2);
        writeln('Custo total      : ',custo_viagem:10:2);
        writeln('Custo acumulado  : ',custo_total:10:2);
        writeln('============================');
        tecla:=readkey;
      end;
      primeiro:=false;
      clrscr;
      writeln('============================');
      writeln('Veiculo numero   : ',n_vel:10);
      writeln('----------------------------');
      writeln('Capacidade       : ',ponteiro^.buffer^.Cap:10:2);
      writeln('Velocidade       : ',ponteiro^.buffer^.Vs:10:2);
      writeln('Custo fixo       : ',ponteiro^.buffer^.Cf:10:2);
      writeln('Custo operacional: ',ponteiro^.buffer^.Co:10:2);
      writeln('----------------------------');
      descricao:=ponteiro;
      buffer:=distancia(ponteiro,ponteiro^.posterior);
      custo_total:=custo_total+ponteiro^.buffer^.Cf+
                   buffer*ponteiro^.buffer^.Co;
      custo_viagem:=ponteiro^.buffer^.Cf+
                    buffer*ponteiro^.buffer^.Co;
      demanda:=0;
      duracao:=0;
    end
    else
    begin
      buffer:=distancia(ponteiro,ponteiro^.posterior);
      custo_total:=custo_total+buffer*descricao^.buffer^.Co;
      custo_viagem:=custo_viagem+buffer*descricao^.buffer^.Co;
      if not(ponteiro^.buffer^.base) then
      begin
        demanda:=demanda+ponteiro^.buffer^.Dem;
        duracao:=duracao+
                 ponteiro^.buffer^.Dem/ponteiro^.buffer^.Tc+
                 ponteiro^.buffer^.Dem/Tcbase;
      end;
      duracao:=duracao+buffer/descricao^.buffer^.Vs;
    end;
    ponteiro:=ponteiro^.posterior;
  until ponteiro=guia;
  writeln('Demanda atendida : ',demanda:10:2);
  writeln('Duracao total    : ',duracao:10:2);
  writeln('Custo total      : ',custo_viagem:10:2);
  writeln('Custo acumulado  : ',custo_total:10:2);
  writeln('============================');
end;
```

A rotina principal carrega a solução inicial, executa o processo de melhoria, procura eliminar veículos ociosos da frota e apresenta a solução melhorada

```
var
  h,m,s,c      : word;
  tempo        : real;
  hh,mm,ss,cc  : real;
begin
  gettime(h,m,s,c);
  tempo:=h*3600+m*60+s+c/100;
  le_arquivo;
  reagrupar;
  elimina_veiculos_inuteis;
  gettime(h,m,s,c);
  tempo:=h*3600+m*60+s+c/100-tempo;
  hh:=int(tempo/3600);
  tempo:=tempo-hh*3600;
  mm:=int(tempo/60);
  tempo:=tempo-mm*60;
  ss:=int(tempo);
  tempo:=tempo-ss;
  cc:=int(tempo*100);
  mostrar_solucao;
  writeln('Tempo de Processamento :
',hh:3:0,':',mm:3:0,':',ss:3:0,':',cc:3:0);
end.
```

IV) GERADOR DE PROBLEMAS ALEATÓRIOS

O gerador de problemas aleatórios cria um conjunto de pontos de demanda aleatórios em um arquivo (NOS.CMP), seguindo parâmetros fornecidos pelo usuário.

Devem ser fornecidos:

- Uma raiz para o gerador de números aleatórios (rand);
- O número de pontos de demanda (n_nos);
- A distância media dos pontos de demanda ao centro geométrico de seu conjunto (raio);
- A demanda média dos pontos de demanda (med_dem);
- O desvio padrão da demanda dos pontos de demanda (sd_dem);
- A taxa de carregamento média dos nós (med_tc);
- O desvio padrão da taxa de carregamento dos nós (sd_tc);
- A posição relativa da base ao centro geométrico do conjunto de pontos de demanda (base, sendo base=0 se a base está no centro geométrico dos pontos de demanda, e base=4 se a mesma está a 2.raio.$\sqrt{2}$);
- A duração máxima de jornada (tv);

A saída é um arquivo, de (n_nos+1) linhas de quatro registros reais, contendo todos os nós do problema a ser resolvido.

A primeira linha possui informações da base da frota, o primeiro registro fornecendo a coordenada X, o segundo a coordenada Y, o quarto a

taxa de carregamento da base. O terceiro registro fornece a duração máxima de jornada do problema.
As demais linhas possuem informações dos pontos de demanda da frota (uma linha para cada ponto de demanda). O primeiro registro fornece a coordenada X, o segundo a coordenada Y, o terceiro a sua demanda e o quarto a sua taxa de carregamento.

Definição de registros

```
uses printer;
type
  nos = record
          X   : real;
          Y   : real;
          Dem : real;
          Tc  : real;
        end;
```

Definição de variáveis globais

```
var
  i,
  ver          : integer;
  ponto        : string;
  no           : array[0..100] of nos;
  rand         : longint;
  n_nos        : integer;
  raio         : real;
  med_dem,
  sd_dem       : real;
  med_tc,
  sd_tc        : real;
  base         : integer;
  Tv           : real;
  r1,r2        : real;
  novo         : boolean;
  arquivo      : file of nos;
```

O procedimento polar_retangular transforma um ponto definido em um sistema de coordenadas polar para coordenadas retangulares

```
procedure polar_retangular(teta,raio:real;var X,Y:real);
begin
  X:=round(Raio*cos(teta));
  Y:=round(Raio*sin(teta));
end;
```

A função normal obtém um valor através de uma função normal, com média e desvio padrão fornecidos

```
function normal(media,desvio:real):real;
var
  z:real;
begin
  if not(novo) then
  begin
    r1:=random;
    r2:=random;
  end;
  if novo then
  begin
    z:=sqrt(-2*ln(r2))*cos(2*pi*r1);
    novo:=false;
  end
  else
  begin
    z:=sqrt(-2*ln(r1))*cos(2*pi*r2);
    novo:=true;
  end;
  normal:=z*desvio+media;
end;
```

A rotina principal gera um conjunto de pontos de demanda em um arquivo seguindo parâmetros fornecidos

```
begin
  if paramcount<9 then
  begin
    writeln('CRIA_NOS rand n_nos raio med_dem sd_dem med_tc sd_tc base
             tv');
    exit;
  end;
  novo:=false;
  val(paramstr(1),rand,ver);
  val(paramstr(2),n_nos,ver);
  val(paramstr(3),raio,ver);
  val(paramstr(4),med_dem,ver);
  val(paramstr(5),sd_dem,ver);
  val(paramstr(6),med_tc,ver);
  val(paramstr(7),sd_tc,ver);
  val(paramstr(8),base,ver);
  val(paramstr(9),Tv,ver);
  Randseed:=rand;
  no[0].X:=base/2*sqrt(2)*raio;
  no[0].Y:=0;
  no[0].Dem:=Tv;
  no[0].Tc:=trunc(normal(med_tc,sd_tc));
  writeln(no[0].X:10:2,no[0].Y:10:2,no[0].Dem:10:2,no[0].Tc:10:2);
  for i:=1 to n_nos do
  begin
```

```
    polar_retangular(random(359),random(trunc(sqrt(2)*raio)),
                     no[i].X,no[i].Y);
    no[i].Dem:=trunc(normal(med_dem,sd_dem));
    if no[i].Dem<1 then no[i].Dem:=1;
    no[i].Tc:=trunc(normal(med_tc,sd_tc));
    if no[i].Tc<1 then no[i].Tc:=1;
    writeln(no[i].X:10:2,no[i].Y:10:2,no[i].Dem:10:2,no[i].Tc:10:2);
  end;
  assign(arquivo,'NOS.CMP');
  rewrite(arquivo);
  for i:=0 to n_nos do
    write(arquivo,no[i]);
  close(arquivo);
end.
```