GUILHERME GUIDOLIN DE CAMPOS

RESOLUÇÃO DE UM PROBLEMA DE ABASTECIMENTO COM AUXÍLIO DE META-HEURÍSTICAS E COMPUTAÇÃO PARALELA

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do Diploma de Engenheiro de Produção

São Paulo
2004

GUILHERME GUIDOLIN DE CAMPOS

RESOLUÇÃO DE UM PROBLEMA DE ABASTECIMENTO COM AUXÍLIO DE META-HEURÍSTICAS E COMPUTAÇÃO PARALELA

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do Diploma de Engenheiro de Produção

Orientador:
Prof. Dr. Hugo T. Y. Yoshizaki

São Paulo
2004

FICHA CATALOGRÁFICA

Campos, Guilherme Guidolin de
Resolução de um problema de abastecimento com auxílio de meta-heurísticas e computação paralela / Guilherme Guidolin de Campos. – São Paulo, 2004.
104 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1. Administração de materiais 2. Distribuição física do estoque 3. Algoritmos genéticos 4. Programação paralela I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II. t.

AGRADECIMENTOS

Ao professor Hugo por toda a ajuda e orientação desde a definição do tema deste trabalho até a revisão final do mesmo.

Ao professor José Carlos Vaz pela orientação nas etapas iniciais do trabalho e à professora Débora Ronconi pelo direcionamento e conselhos importantes.

À Patrícia Belfiore pela ajuda no entendimento do problema e levantamento de dados e informações sobre a empresa.

Ao Prof. Kazuo Nishimoto e ao Antônio Russo pela disponibilização das instalações do Cluster da Naval para a execução dos experimentos computacionais e ao Gabriel Winckler pela ajuda na operação do mesmo.

À Cintia Celidônio pela paciência, incentivo e auxílio fundamental na elaboração deste trabalho.

À toda minha família e amigos que em algum momento me aconselharam e incentivaram, contribuindo e tornando possível a realização deste trabalho

RESUMO

O presente trabalho propõe a utilização de meta-heurísticas e computação paralela para a resolução do problema de roteamento de veículos no processo de abastecimento das unidades de um grupo varejista com restrições operacionais. O problema consiste na determinação de um conjunto de rotas econômicas que devem atender a necessidade de abastecimento de cada uma das lojas do grupo respeitando todas as restrições, principalmente janelas de tempo, duração da jornada e frota heterogênea. A estratégia adotada para a resolução do problema consiste na utilização de uma adaptação da heurística construtiva proposta por Clarke & Wright (1967) como solução inicial. Posteriormente são utilizados alguns algoritmos mais sofisticados buscando-se melhorias, dentre eles o algoritmo genético paralelo resolvido com o auxílio de um cluster de computadores. Os resultados obtidos demonstram que a heurística construtiva básica apresenta bons resultados para o problema, mas ainda pode ser melhorada com o uso das técnicas mais sofisticadas. A aplicação dos métodos propostos, proporcionou uma redução no custo total da operação da ordem de 11% comparando-se com as soluções utilizadas originalmente pela empresa.

ABSTRACT

The present work considers the utilization of meta-heuristics and parallel computing to solve the vehicle routing problem in the branches supply process of a retail group with operational constraints. The problem is about finding a set of economic routs which must meet the supplying needs of each of the group's branches respecting all the constraints, especially time windows, rout length and heterogeneous fleet. The adopted strategy for the problem solution consists in the utilization of an adaptation of the constructive heuristics proposed by Clarke & Wright (1967) as initial solution. Then some more sophisticated algorithms are applied in order to achieve improvements, such as parallel genetic algorithms supported by a cluster of computers. The results obtained show that the basic constructive heuristics presents good results for the problem, but it still can be improved by applying more sophisticated techniques. The use of the proposed methods, provided about 11% reductions in the total cost of operations when compared to the original solutions applied by the company.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

INTRODUÇÃO

Todo sistema logístico tem como objetivo proporcionar aos clientes os bens desejados no local, tempo e quantidade que melhor atendam às suas necessidades, incorrendo no menor custo possível. Para tanto, as empresas devem tomar uma série de decisões que envolvem principalmente a estratégia de localização de suas fábricas e centros de distribuição, estratégia de estoque envolvendo previsão de demanda, compras, políticas de armazenagem, etc. e estratégia de transporte. Uma vez definidas as principais estratégias logísticas da empresa e estabelecida sua estrutura física, cada atividade deve ser planejada de forma a se atingir um nível de excelência operacional que garanta o uso eficiente dos recursos disponíveis oferecendo o melhor nível de serviço pelo menor custo.

Segundo Ballou (2001), dentre todas as atividades logísticas, o transporte é aquela que absorve a maior parcela dos custos. A seleção do modal de transporte a ser utilizado, a definição de políticas de entrega, consolidação das cargas e roteirização, e programação dos veículos estão entre as atividades mais importantes a serem executadas neste campo. Quanto maior o número de produtos diferentes com que uma empresa trabalha, menores os prazos e maior o número de pontos de entrega, mais complexa se tornam estas atividades.

A empresa em questão é um destes casos onde a complexidade da atividade logística faz com que esta seja uma preocupação central. Assim, estaremos propondo neste trabalho alguns modelos para determinar de forma estruturada, maneiras melhores de se transportar as mercadorias dos centros de distribuição para as lojas do grupo. Buscaremos assim, diminuir os custos da atividade de transporte e melhorar o nível de serviço, medido pelo número de restrições que são violadas, tais como o atendimento total da demanda, horários de entrega, entre outras.

1. DESCRIÇÃO DA EMPRESA E DEFINIÇÃO DO PROBLEMA

Neste capítulo descreveremos a empresa estudada e definiremos o problema a ser resolvido no decorrer do trabalho. Num primeiro momento detalharemos a estrutura da empresa e seus processos, dando uma visão geral da mesma. Na segunda parte será definido o problema a ser solucionado e os objetivos a serem alcançados pelo presente trabalho.

Para preservar em sigilo as informações consideradas estratégicas pela empresa, e de acordo com sua própria solicitação, o nome da companhia não será revelado, assim como alguns dados ao longo deste trabalho serão modificados.

É importante deixar claro neste momento que o autor não realizou seu estágio nesta companhia como é de costume na elaboração dos Trabalhos de Formatura. No entanto, a escolha do tema, a definição do problema a ser abordado, e a troca de informações necessárias para a elaboração deste trabalho foram feitas de comum acordo com a gerência da empresa responsável pelo processo logístico. Além disso, o tema escolhido para o trabalho apresenta forte relação com as duas iniciações científicas realizadas pelo autor ao longo dos últimos anos nas áreas de logística e de computação paralela.

1.1. A EMPRESA

O Grupo é um dos pioneiros no setor varejista do Brasil. Atuando a décadas, é hoje um dos maiores grupos no varejo com grande participação em um mercado altamente fragmentado. Ao longo de sua história o grupo cresceu constantemente de forma orgânica e através de aquisições. Atualmente, além de suas operações comerciais, a empresa patrocina diversas atividades sociais

e culturais.

A Cadeia de Suprimentos, foco deste trabalho, é uma área estratégica que engloba as funções logísticas e comerciais de todo o grupo. A estrutura logística é responsável pelo abastecimento de produtos às lojas, desde a gestão dos estoques até o fluxo físico de mercadorias. Ela é capaz de agregar valor à operação melhorando a receita, reduzindo a ruptura (falta de produtos nas lojas) e diminuindo custos de transportes e investimentos em estoque.

A Companhia opera diversos tipos e tamanhos de lojas com eficiência, graças a uma estrutura centralizada, em forma de Centros de Distribuição (CDs). O Grupo ultrapassou o patamar de 80% de centralização dos estoques em 2003, um percentual próximo aos padrões internacionais. Outro benefício da centralização foi a redução nos investimentos em estoque, por conta do aumento do giro dos produtos, além do maior controle sobre o processo de abastecimento das lojas.

A estrutura logística do grupo é formada por 10 CDs, que totalizam uma capacidade de armazenagem superior a 200 mil metros quadrados de área construída. A companhia opera CDs multi-categoria, que atendem a determinadas regiões com raio de atuação de até 500 quilômetros em 6 capitais brasileiras. Em São Paulo, possui quatro CDs especializados em categorias específicas, dado o tamanho do mercado que justifica esta segmentação.

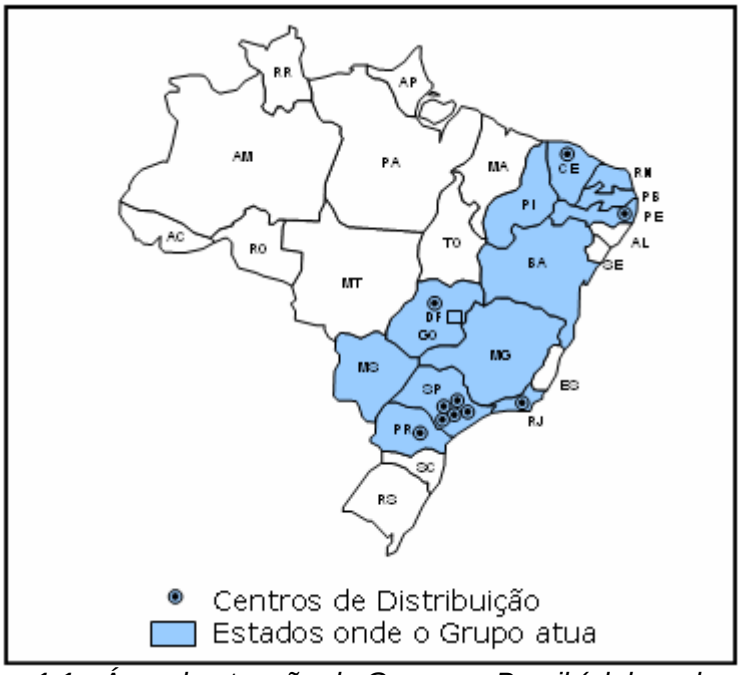

Figura 1.1 - Área de atuação do Grupo no Brasil (elaborado pelo autor)

Na próxima seção iremos detalhar a forma como estes centros de distribuição realizam o abastecimento das lojas do Grupo e como ocorre este processo, desde o recebimento dos pedidos até a entrega efetiva.

1.1.1. Descrição do Processo de Abastecimento

De acordo com suas necessidades, os gerentes de loja ou seção fazem diariamente seus pedidos. A entrega pode ser feita de duas formas: via depósito ou diretamente pelo fornecedor com o pedido sendo feito via EDI (Electronic Data Interchange). Após o fechamento do faturamento uma Central de Programação (CP) decide como será feita a distribuição das cargas do CD às lojas através de roteirização, de acordo com os veículos disponíveis. A separação das cargas solicitadas pela CP é feita pela parte operacional dos CDs. Caso a carga a ser entregue exceda a capacidade dos veículos, é feita uma revisão, ou seja, estuda-se a possibilidade de alterar o tipo modelo do veículo ou combinar e alterar os roteiros. O objetivo final é atender aos pedidos

dos clientes nos prazos adequados com o pleno atendimento da demanda e sem que nenhuma restrição seja violada.

A apresenta uma simplificação dos fluxos dos processos que ocorrem na Central de Programação.

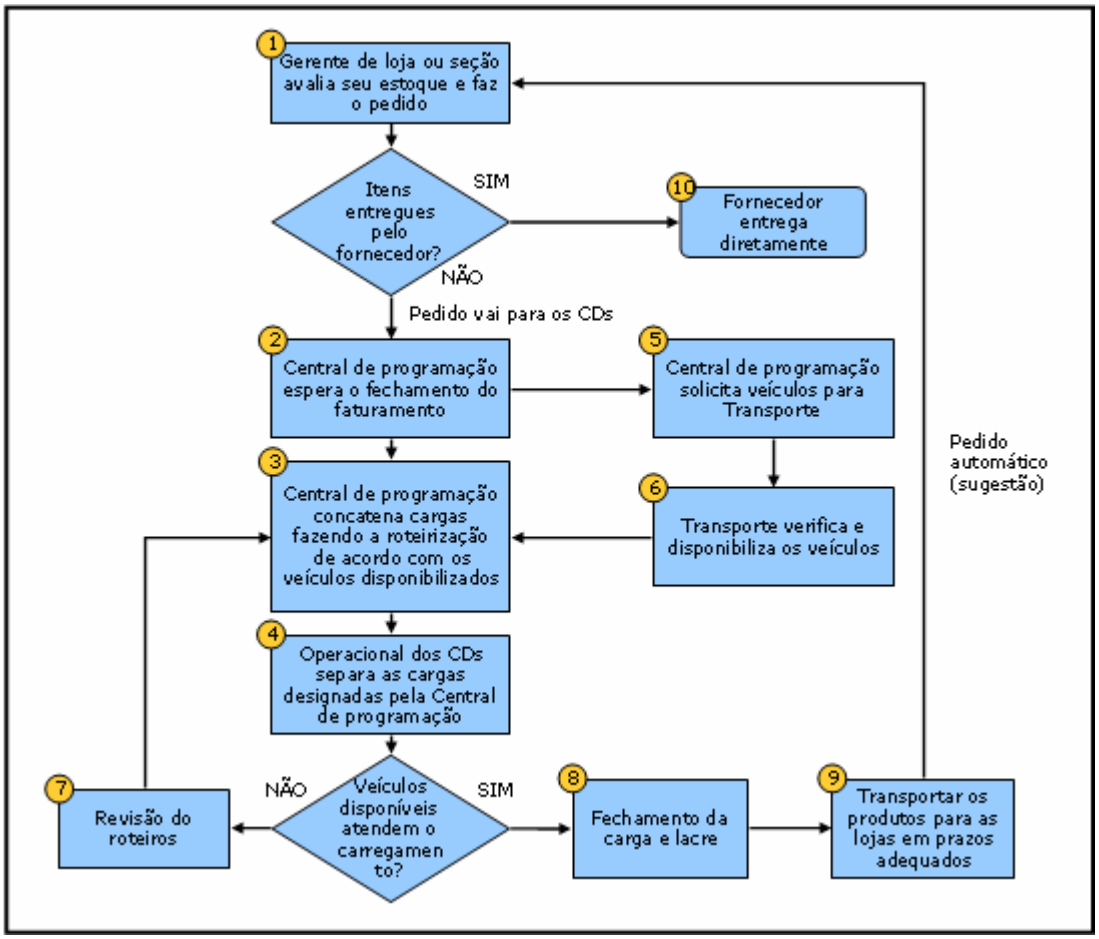

Figura 1.2 - Fluxograma da operação de abastecimento (elaborado pelo autor)

Todo o processo de abastecimento tem início com o pedido feito pelo gerente das lojas que avalia o estoque disponível para todos os itens vendidos e comunica a Central de Programação sobre sua necessidade. Em algumas situações, dependendo do tipo de produto e da quantidade solicitada, o pedido

segue diretamente para o fornecedor via EDI que se responsabiliza pela entrega para a loja.

Há um algoritmo que gera esses pedidos automaticamente, porém os dados são sistematicamente alterados pelo gerente/chefe da seção. O algoritmo leva em conta apenas os estoques nas lojas. Algumas lojas consideram uma questão logística, o arredondamento de paletes, já outras não interferem nos pedidos e são lojas que têm menos ruptura.

Uma vez finalizados os pedidos, estes são encaminhados para a Central de Programação que inicia o processo de formação de cargas após o fechamento do faturamento. Em seguida são formados os roteiros que serão utilizados na entrega dos produtos para as lojas.

Atualmente, o processo de formação de cargas e roteirização utiliza alguns itinerários determinados anteriormente que são ligeiramente adaptados de acordo com a demanda específica do dia. O cálculo está vinculado a parâmetros de equipamentos de movimentação, ocupação do veículo e capacidade física da loja. A formação da carga segue os seguintes parâmetros cadastrados no sistema:

- Cadastro de loja: tipo de veículo, tipo de equipamento, prioridade de entrega;
- Cadastro de CD para encaixe: informar quais CDs podem realizar encaixes;
- Peso/m^3 por equipamento/Loja: paletes, gaiolas, rolltainers;
- Cadastro por veículo: capacidade máxima por veículo/região (capital, interior, interestadual), em peso e m^3;
- Cadastro de itinerários: cadastro de agrupamento de lojas de acordo com o tipo de veículo;

Depois de efetuado o processamento do faturamento, o sistema segue as rotinas, considerando sempre o maior tipo de veículo e liberando para usuário as cargas com 90% da ocupação. Neste processo são realizadas as seguintes etapas:

1) Com base nos parâmetros, o melhor veículo é escolhido para 1 entrega, ou seja, 1 loja;
2) Se houver cadastro de CDs para encaixe, o sistema otimiza a carga das duas ou mais categorias para 1 entrega (1 loja);
3) Se os encaixes não proporcionarem a otimização máxima da carga, o sistema buscará a composição de acordo com o agrupamento de itinerários;
4) Se não houver possibilidade de encaixe, o sistema formará cargas de acordo com os agrupamentos de itinerários;
5) Finalmente, busca-se a otimização com veículo alternativo. Caso não seja possível, aguarda-se o próximo faturamento.

Este procedimento realizado pelo sistema tem a grande vantagem de ser muito rápido e bastante intuitivo para os operadores. No entanto, ele é relativamente fraco quanto à otimização dos roteiros, uma vez que não considera diferentes combinações de lojas nem seqüências alternativas para serem incluídas em cada um deles.

A atividade de transporte é conduzida por terceiros, uma vez que o Grupo tomou a decisão de não possuir veículos próprios, dada a dificuldade de se gerenciar de forma eficiente uma frota do tamanho necessário. O transporte é feito por 40 transportadoras terceirizadas, que mantêm cerca de 500 veículos dedicados ao abastecimento das lojas do Grupo. A média é de mil viagem por dia, tendo sido atingido o recorde de mais de 3 mil viagens numa única data.

1.2. DIAGNÓSTICO DO PROBLEMA

Com uma operação do porte mencionado, há anos o Grupo investe em sistemas de previsão de demanda e controle de estoque para atender de maneira adequada seus clientes. Por outro lado, o procedimento de abastecimento das lojas é nitidamente deficiente. Tal fato ocorre devido à complexidade inerente ao problema e à inexistência de ferramentas adequadas de otimização. Vamos explicar um pouco melhor este problema a seguir.

No final de cada dia, é calculada a necessidade de abastecimento de cada uma das lojas. Nesta etapa do processo já surgem complicações, pois algumas lojas da rede funcionam num sistema 24h. Ao mesmo tempo, os centros de distribuição fazem a contabilização de todas as movimentações que ocorreram durante o dia (entregas para as lojas e recebimento de fornecedores), e verificam a disponibilidade de cada um dos itens em estoque. O sistema então prioriza os pedidos de acordo com uma série de critérios pré-definidos pela direção, gerando uma lista de tudo que deverá ser entregue no dia seguinte. O número total de itens a serem entregues ultrapassa uma dezena de milhares.

As lojas trabalham com um conceito chamado de janela de recebimento que consiste num intervalo de tempo no qual o abastecimento das mesmas deve ser realizado. Além disso, o volume elevado de carga transportada exige que o processo de entregas comece cedo, para garantir que todas as lojas sejam abastecidas num período adequado. Desta forma, o tempo disponível entre o final do processo de priorização do abastecimento e o início das entregas é muito reduzido, da ordem de uma hora. Este é o tempo que os funcionários têm para planejar todas as rotas e decidir que tipo de caminhão irá percorrer cada uma delas atendendo a totalidade das lojas nos horários adequados.

Uma vez que os roteiros tenham sido planejados, tem início o processo de

entrega para as lojas a partir do centro de distribuição. A ilustra de maneira simplificada um possível resultado da roteirização. Neste caso, a demanda das 5 lojas será entregue através de 2 roteiros, sendo cada um deles percorrido por um veículo diferente.

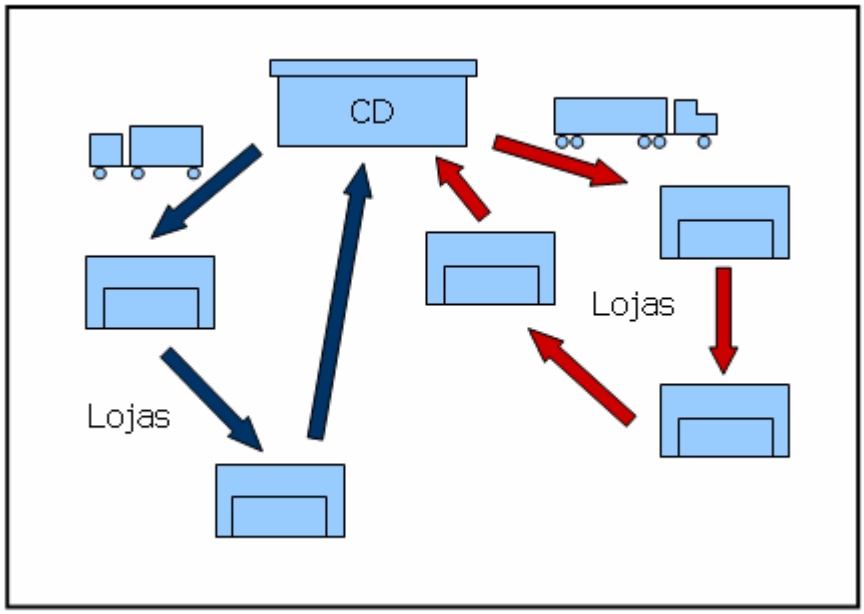

Figura 1.3 - Processo de abastecimento simplificado (elaborado pelo autor)

Como podemos ver, esta é uma tarefa enorme e o tempo disponível inadequado. Assim, a primeira solução encontrada que atenda todas as restrições do problema (que abordaremos em profundidade mais adiante) é utilizada, mesmo que não seja a melhor do ponto de vista da otimização dos recursos.

Neste sentido, o objetivo final deste TF consiste na elaboração de um modelo para a seleção de veículos, consolidação das cargas e roteirização que atenda as restrições enfrentadas no abastecimento das lojas, e implementar métodos de solução que consigam obter resultados melhores do que os atuais no período de tempo disponível.

O resultado esperado deste modelo é que ele permita construir diariamente um conjunto de roteiros especificando todas as lojas que serão atendidas, a seqüência de atendimento e o tipo de veículo que atenderá cada um deles, como na . Caso se consiga realizar estes objetivos a contento, o Grupo vai estudar a possível implementação da solução proposta através de uma ferramenta.

Como o problema abordado é extremamente complexo do ponto de vista de otimização, como veremos adiante, um último objetivo deste trabalho é a aplicação da Computação Paralela através de um Cluster de computadores na resolução do mesmo, visando avaliar o potencial desta ferramenta.

O estudo será focado no atendimento das lojas localizadas no estado de São Paulo a partir do principal centro de distribuição da empresa, o que representam um total de 323 unidades a serem atendidas. Como nem todas as lojas devem ser abastecidas todos os dias da semana, escolhemos um dia típico para utilizar como base para o trabalho. Neste dia, 214 lojas registraram pedidos para o CD em questão. O escopo deste trabalho também estará restrito à cargas paletizadas, que podem ser transportadas sem o uso de caminhões especiais e que compõe a grande maioria das entregas feitas no processo de abastecimento. Da mesma forma, não serão considerados os pedidos atendidos diretamente pelos fornecedores, mas apenas aqueles destinados aos CDs.

2. REVISÃO DA LITERATURA

Esta revisão da literatura visa fornecer subsídios à definição clara do problema e identificação dos melhores métodos disponíveis para sua solução. Nela discutiremos os problemas de transporte e como o problema de roteirização de veículos se enquadra nesta categoria, bem como os critérios para a classificação destes problemas. Na segunda parte buscaremos nos aprofundar nos métodos de solução existentes para os problemas de roteirização de veículos. Em seguida será feita uma introdução sobre a computação paralela e seu uso na solução de problemas complexos. Finalmente será apresentado um método de solução que combina os algoritmos apresentados com a computação paralela.

2.1. PROBLEMAS DE TRANSPORTE

Segundo Ballou (2001), embora haja uma grande variedade de problemas de distribuição, podemos agrupá-los em alguns tipos básicos. Há o problema de encontrar um caminho através de uma rede onde o ponto de destino é diferente ao ponto de origem. Há um problema similar onde ocorrem múltiplos pontos de origem e destino e o problema de roteamento quando os pontos de origem e destino são exatamente os mesmos ().

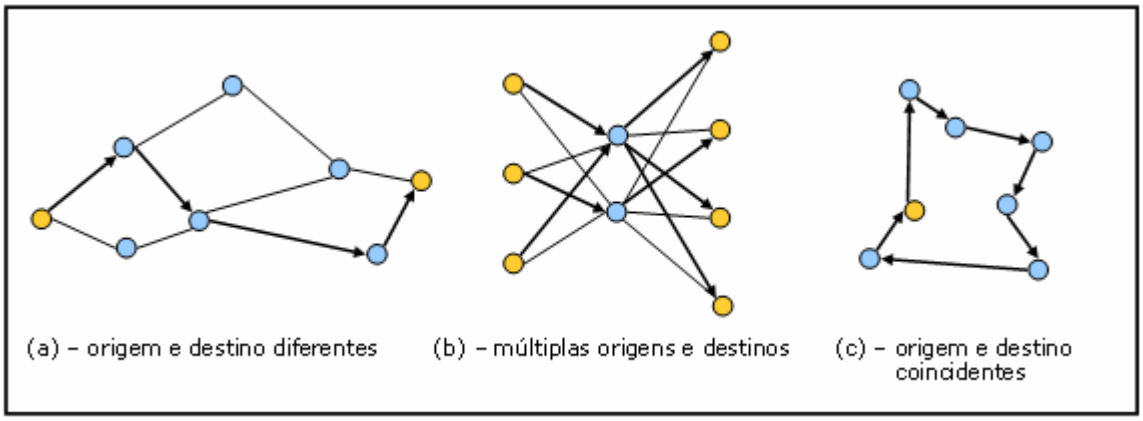

Figura 2.1 - Três problemas clássicos de distribuição (elaborado pelo autor)

O problema a ser tratado no presente trabalho enquadra-se nesta terceira categoria, onde se encontram alguns dos problemas clássicos de transporte como o PRV (Problema de Roteamento de Veículos) e o PCV (Problema do Caixeiro Viajante), sendo que em ambos os casos os pontos de origem e destino são coincidentes. A diferença entre eles é que no primeiro, múltiplas rotas podem ser formadas para percorrer todos os pontos, enquanto no segundo eles devem ser atendidos por apenas uma rota.

Neste capítulo discorreremos sobre os problemas de transporte e apresentaremos a definição e alguns conceitos sobre os problemas de roteamento de veículos. Em seguida, apresentaremos alguns modelos tradicionais utilizados para o tratamento deste tipo de problema e novos métodos considerados na sua solução.

2.1.1. Problema do Caixeiro Viajante – PCV

O problema do caixeiro viajante pode ser considerado uma forma mais simples do problema de roteamento de veículos e, portanto, sua compreensão será útil na busca de um método de solução eficiente para o mesmo. Na prática, o PCV pode ser utilizado para representar cada um dos roteiros no processo de abastecimento, como os da Este é um problema de otimização associado à determinação dos caminhos ótimos sobre um grafo iniciando e terminando no mesmo vértice sem nunca repetir uma visita. Assim, o objetivo do PCV é encontrar em um grafo $G = (N, A)$ o caminho de menor custo.

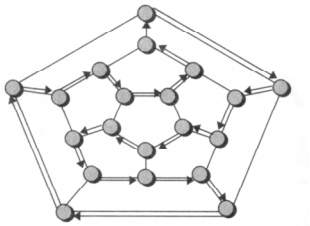

Figura 2.2 - Grafo de um PCV e sua solução (extraído de Goldbarg, Luna, 2000)

Existem diversas formulações para este problema. Dantzig, Fulkerson e Johnson (1979) formularam o PCV como um problema de programação binária sobre um grafo *G* = (*N*, *A*), como segue:

$$\text{Minimizar } z = \sum_{j=1}^{n}\sum_{i=1}^{n} c_{ij}\, x_{ij} \qquad (1)$$

sujeito a:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall_j \in N \qquad (2)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall_i \in N \qquad (3)$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset N \qquad (4)$$

$$x_{ij} \in \{0,1\} \qquad \forall_{i,j} \in N \qquad (5)$$

Onde a variável binária x_{ij} assume valor igual a 1 se o arco $(i,j) \in A$ for escolhido para integrar a solução, e 0 em caso contrário, c_{ij} é um custo associado ao arco (i,j), e *S* é um subgrafo de *G* em que $|S|$ representa o número de vértices desse subgrafo. As restrições (2) e (3) garantem que cada ponto da rede tenha um e somente um arco chegado e um arco saindo. O conjunto de restrições (4) impede o surgimento de sub-rotas na solução ótima.

Problema do caixeiro viajante com janela de tempo - PCVJT

Um dos problemas relacionados ao PCV é o Problema do Caixeiro Viajante com Janela de Tempo - PCVJT. Ao associar uma variável t_{ij} a cada arco do grafo G = (N, A), representando a duração do percurso entre os nós i e j, pode-se imaginar que a chegada a cada vértice *i*, $i \in N$ do grafo G é restrita ao intervalo [a_i, b_i], denominado de janela de tempo. Assim, são consideradas viáveis apenas as soluções que atendem a restrição de janela de tempo. Este modelo é mais parecido com o problema que estamos tentando resolver, pois conforme visto no item 1, as lojas operam com janela de tempo.

2.1.2. Problemas de Roteamento de Veículos

Para que possamos compreender melhor os problemas de roteamento de veículos (PRV), é necessário definir os sistemas de roteamento. Segundo Goldbarg e Luna (2000), considera-se um sistema de roteamento um conjunto organizado de meios com o objetivo de atender pontos de demanda localizados em arcos ou vértices de alguma rede de transportes.

Na operação de abastecimento, é necessário um plano efetivo e flexível de entregas, de modo a atender às especificações referentes ao nível de eficiência do serviço de transporte. Este plano ou roteiro deve ser definido quantitativamente e atender da melhor maneira possível todas as restrições do problema. Dentro desse contexto, surge um grupo de problemas de característica combinatória e de grande dificuldade de solução, que se denominam problemas de roteamento de veículos. O objetivo do planejamento será estabelecer um roteamento e sequenciamento, e o emprego de veículos que conduzam à minimização do custo da atividade. Segundo Goldbarg e Luna (2000), a idéia básica do problema de roteamento é, com o uso de veículos, visitar uma série de clientes ao menor custo possível, atendendo a todas as demais imposições do problema.

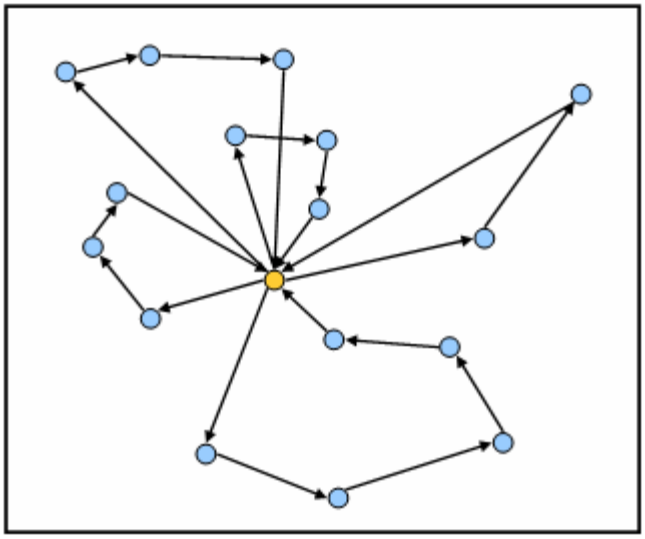

Figura 2.3 - Exemplo de formação de rotas para um depósito (elaborado pelo autor)

2.1.3. Variações dos Problemas

Apesar do conceito de roteamento de veículos não ser de difícil compreensão, existe um grande número de variações do mesmo. Ronen apud Cunha (1997) afirma que os problemas pertencentes a esta classe podem ser agrupados em três categorias básicas:

- Problemas relativos ao transporte de passageiros: programação de linha de ônibus, sistemas de táxi, transporte de idosos e deficientes (dial-a-ride), transporte escolar, entre outros;
- Problemas de prestação de serviços: equipes de reparos, coleta de lixo, entrega postal, serviço de limpeza de vias, etc;
- Problemas de transporte de cargas: entrega e coleta de produtos em múltiplas localidades.

O problema de transporte de carga pode ser classificado ainda segundo uma série de critérios que englobam, entre outros:

- Tamanho, composição, e estrutura de custos da frota;
- Número de bases de origem e destino para os veículos;
- Demanda (entrega ou coleta), determinística ou estocástica;
- Atendimento total ou parcial da demanda;
- Aleatoriedade dos locais e horários de entrega e coleta;
- Limites de distância ou duração dos roteiros.

Assad (1988) propõe a classificação dos problemas de roteirização de veículos segundo alguns aspectos básicos do problema:

Aspectos do problema	**Variações**
Demanda	- entrega, coleta ou backhaul - tipos diferentes de produtos - atendimento total ou parcial - prioridade de clientes - contratação de serviços de terceiros - demanda conhecia ou aleatória - entregas periódicas ou única
Frota de Veículos	- homogenia ou heterogênea - restrição de capacidade / carregamento - vínculo entre veículo e base - compatibilidade ente veículo e produto - número fixo ou variável de veículos - veículo em um ou múltiplos depósitos
Pessoal	- duração da jornada de trabalho - opção de horas extras - número fixo ou variável de motoristas - hora e local do início da jornada - paradas (almoço, descanso) - possibilidade de viagens maiores que 1 dia

Programação	- janelas de tempo (rígidas ou flexíveis) - tempo de carga e descarga - horário de abertura e fechamento - dias da semana para o atendimento de clientes
Informações	- disponibilidade de dados geográficos e redes viárias - recursos de localização de endereços - tempos de viagem - localização dos veículos - informações sobre o crédito dos clientes

Tabela 2.1 - Diferentes aspectos de um problema de roteirização (adaptado de Assad 1988)

Dada a grande variedade de aspectos do problema de roteirização de veículos, fica evidente a dificuldade de se estabelecer uma classificação única para todos os problemas. Na prática, cada caso poderá assumir quaisquer combinações destes fatores tornando-se quase único. No entanto existem alguns casos mais gerais cuja compreensão nos ajudará no esforço de encontrar o método mais adequado de solução para o problema específico que estaremos abordando neste trabalho.

Problema de Roteamento de Veículos (PRV): Tem como objetivo encontrar um conjunto de rotas, iniciando e terminando em um mesmo ponto, de forma a minimizar a distância total percorrida e/ou o número de veículos utilizados. A formulação deste problema inclui basicamente as seguintes restrições:

- Nenhum veículo deve sair do ponto inicial mais que uma vez;
- Todos os veículos devem retornar ao ponto de origem;
- Todos os destinos devem ser visitados uma única vez.

Problema de Roteamento de Veículos com Capacidade (CPRV): o PRV com capacidade obedece à mesma formulação do problema original com a inclusão

de uma restrição de volume transportado nas rotas limitado pela capacidade de carga do veículo. Neste problema, a cada ponto da rede é atribuída uma demanda que deverá ser atendida pela rota a que ele pertença.

- O total das entregas feitas por um veículo não pode exceder sua capacidade;
- A demanda de todas as lojas deve ser atendida.

Problema de Roteamento de Veículos com Janela de Tempo (PRVJT): Trata-se uma generalização do PRV. A solução do PRVJT deve garantir que o tempo de coleta e entrega do usuário não viole a restrição de janela de tempo. Este problema esta sujeito às restrições originais do PRV além de:

- Os tempos máximo e mínimo de viagem de cada veículo devem ser respeitados;
- Os destinos não podem ser visitados após o final da janela de recebimento.

Problema de Roteamento de Veículos com Frota Heterogênea (PRVFH): o PRV com frota heterogênea é um caso mais próximo da realidade, pois considera que os veículos utilizados no roteamento possuem características diferentes. Assim, podemos considerar que tanto os custos fixos de uma frota, quanto os variáveis, como combustível, etc, são diferentes de acordo com o modelo do veículo. Da mesma maneira, a capacidade de carga de cada um deles pode ser diferente, o que torna o problema bem mais complexo. Além de decidir quais as melhores rotas para realizar as entregas devem ser estabelecidos os veículos que irão percorrer cada uma delas.

Problema de Roteamento de Veículos com Frota Fixa (PRVFF): O problema básico de roteamento de veículos considera a existência de um número ilimitado de veículos disponíveis para roteirização. No entanto, muitas vezes as empresas possuem um determinado número de veículos próprios e não tem a

opção conseguir mais veículos. Neste caso, podemos considerá-lo como um problema de roteamento com frota fixa, onde o aproveitamento de cada veículo disponível passa a ser fundamental para garantir o atendimento da demanda.

- O número de veículos utilizados não pode exceder sua disponibilidade

Certamente existem outros casos do PRV que podem ser obtidos pela combinação destas e de outras características do problema, tais como as descritas na . Mesmo assim, a compreensão dos aspectos fundamentais destes problemas básicos apresentados ajudará tanto na especificação do problema a ser resolvido quanto na identificação dos métodos de solução mais apropriados.

A solução do problema PRV e todas as suas variações consiste em uma rota ou em um conjunto de rotas que especifica a seqüência dos destinos que deverão ser visitados. No entanto, para se obter uma boa solução para o problema é necessário investir um considerável tempo no desenvolvimento de um programa computacional apropriado. Quase todos os modelos implementados para o PRV utilizam procedimentos heurísticos, que conseguem obter soluções boas para problemas reais, uma vez que os modelos otimizantes não conseguem chegar a uma solução para classe de problemas dada a sua grande complexidade e característica combinatória. Na próxima seção estaremos analisando os métodos de solução encontrados na literatura e sua aplicabilidade ao problema abordado neste trabalho.

2.2. MÉTODOS DE SOLUÇÃO

Existe na literatura uma grande variedade de métodos utilizados na solução dos problemas de transporte, especialmente o de roteirização de veículos. Um dos fatores que contribui para este fato é a grande dificuldade de se encontrar soluções ótimas para o problema devido a sua alta complexidade e possibilidade de variações. Assim, alguns métodos que conseguem bons

resultados para um grupo de problemas não se saem tão bem para outros com algumas variações em sua formulação. Desta forma, muitos pesquisadores se esforçam no desenvolvimento de novos métodos e aplicações.

A grande maioria dos métodos de solução existentes se enquadra em uma de três categorias: métodos otimizantes, heurísticas construtivas e métodos iterativos de melhoria ou meta-heurísticas. Nesta seção iremos abordar alguns destes métodos com o intuito de estabelecer a melhor estratégia de solução para o problema tratado neste trabalho.

2.2.1. Dificuldade de Solução

Os problemas de roteamento de veículos variam quanto a sua complexidade dependendo do número de variáveis e restrições que o problema considera em sua formulação. Alguns problemas podem ser considerados quanto a sua complexidade como intratáveis. Mesmo com o uso de computadores teríamos dificuldades muito grandes com esses problemas. Não se trata somente de aumentar a capacidade da máquina, pois a dificuldade reside na natureza combinatória desse tipo de problema que, até hoje, tem impedido a concepção de algoritmos eficientes de solução. Esses problemas são tratados como NP-Árduos (do inglês NP-Hard). Em outras palavras, o esforço computacional para a sua resolução cresce exponencialmente com o tamanho do problema, dado pelo número de pontos a serem atendidos. Para esses problemas complexos, na busca por boas soluções, são utilizadas técnicas para alcançar soluções próximas da ótima, como as heurísticas. A , abaixo, mostra como a pesquisa operacional desenvolveu estratégias para tratar cada tipo de problema.

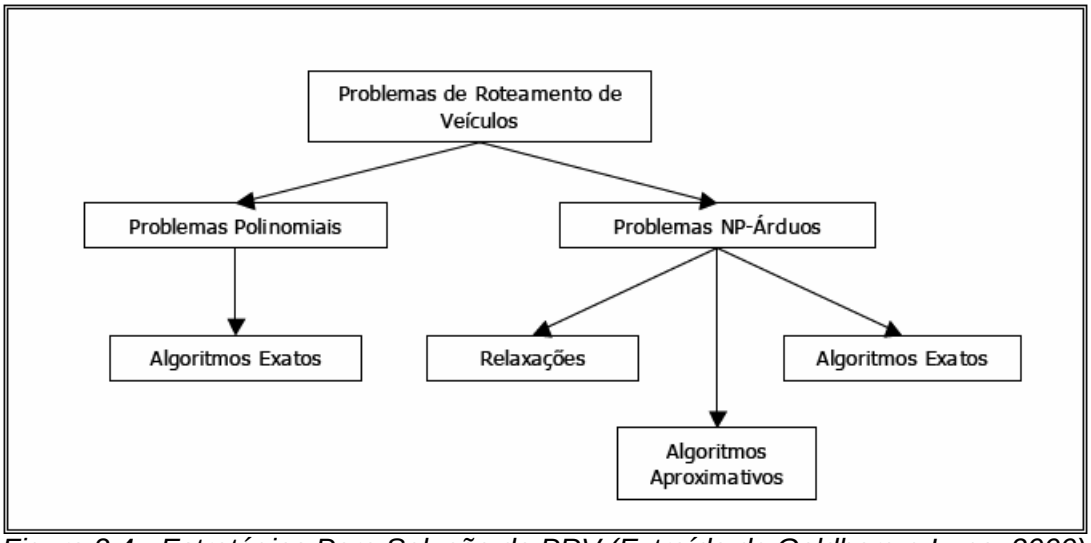

Figura 2.4 - Estratégias Para Solução do PRV (Extraído de Goldbarg e Luna, 2000)

Devido ao seu caráter fortemente combinatório, a maioria dos problemas práticos de roteamento de veículos (PRV) é do tipo NP-Árduo (Goldbarg e Luna, 2000). Como visto na , este tipo de problema poder ser tratado com algoritmos exatos, relaxações e algoritmos aproximativos. Os algoritmos exatos são usados apenas em casos de PRV com poucas variáveis e restrições. Normalmente, este tipo de resolução vem acompanhado de técnicas de relaxações, que ajudam a eliminar algumas variáveis e restrições do problema. Em outros casos, são utilizados algoritmos aproximativos (heurísticas), que buscam de maneira simplificada, mas eficiente, soluções que aproximem ao máximo da solução ótima do problema. A seguir apresentaremos as principais classificações das estratégias e métodos de solução.

2.2.2. Estratégias e Métodos de Solução

As estratégias de solução para roteirização de veículos são classificadas, segundo Bodin apud Cunha (1997), da seguinte forma:

1 – Agrupa e Roteiriza (cluster first – route second) – é o procedimento que primeiramente agrupa os nós de demanda e depois constrói as rotas mais

econômicas em cada um destes grupos. Essa estratégia geralmente é utilizada em problemas básicos com um único deposito na roteirização de veículos.

2 – Roteiriza e Agrupa (route first – cluster second) – incluindo todos os nós de demanda, primeiro constrói-se uma grande rota sem solução possível, depois esta é dividida em rotas factíveis e menores. Este modelo já foi usado na resolução de alguns problemas contendo roteirização com frota heterogênea de veículos e em outros com varredura de ruas.

3 – Economias ou inserções – procedimento onde se constrói uma solução em um determinado caminho, sendo que em cada etapa deste processo se compara a configuração em construção, com uma solução alternativa. A solução alternativa será a que apresentar maior economia em algum critério adotado, como custo total, ou a que inserir na rota em construção, entidades da demanda, de forma menos custosa. Conclui-se o processo quanto todas as inserções factíveis tiverem sido realizadas.

4 – Melhoria / Troca – procedimento heurístico onde há uma troca de arcos em cada etapa do processo, buscando uma nova solução com menor custo, este se repete até que não haja soluções mais econômicas. Este método também é conhecido como r-opt, onde r é o numero de arcos trocados a cada etapa, todos os r arcos são trocados até que não haja Nenhuma troca factível que melhore o custo do problema. Os valores mais usados são r=2 (2-opt) e r=3 (3-opt).

5 – Otimização interativa – para a solução do problema há uma grande interação humana, este tomador de decisões baseia-se em seu conhecimento no modelo de otimização, revisando parâmetros e inserindo correções, aumentando assim a possibilidade de implementação deste método de resolução. Quem adaptou esse método pela primeira vez na solução de problemas com roteirização de veículos foi Krolak.

6 – Procedimentos exatos – os algoritmos exatos em problemas de roteirização e a programação do tipo NP-Hard têm sido limitados, com exceção do problema do caixeiro viajante, que inclui o método de particionamento para a programação de pessoal e algoritmos exatos para a programação de veículos, utilizando técnicas de branch and bound e programação dinâmica.

Cunha (1997) classifica os métodos de solução nas seguintes categorias:

- Métodos exatos - garantem uma solução ótima;
- Métodos heurísticos – não garantem uma solução ótima, mas sub-ótimas, que necessitam de um menor esforço computacional.
- Métodos emergentes – compostos de técnicas mais recentes e avançadas, baseadas em sistemas especializados, métodos de busca ou interativos. Exemplos deste método são as meta-heurísticas como algoritmos genéticos e busca tabu.

A seguir descreveremos sucintamente os principais métodos meta-heurísticos pesquisados.

Simulated annealing (SA): criada por Kirkpatrik, baseia-se em conceitos da mecânica estatística, sendo uma analogia do processo de recozimento dos sólidos. Nos problemas de otimização, os estados correspondem às soluções possíveis, e a energia, a função a ser minimizada. Sendo que os SAs exploram as soluções através de geração seqüencial e aleatória, tendo transição por uma pequena perturbação, as melhores soluções são aceitas e armazenadas. Há também um parâmetro que equivale a temperatura em um sólido, este parâmetro é gradualmente reduzido e com ele diminui a probabilidade de aceitação de soluções piores. Assim sendo, a probabilidade da solução se deteriorar tende a zero conforme cresce o número de iterações.

Algoritmo Genético (AG): proposto por Holland, segue princípios biológicos da reprodução evolutiva, diferente de outras meta-heurísticas, os AGs não exploram soluções seqüencialmente e sim populações de soluções, onde a mais apta será selecionada para as próximas iterações. Os operadores utilizados nos AGs são a reprodução, que copia os indivíduos de uma geração para outra, crossover ou cruzamento, que recombina características, e mutação, que produz pequenas mudanças nos indivíduos garantindo uma diversidade na população e permitindo a exploração de novas regiões.

Busca Tabu (BT): incorpora conceitos de inteligência artificial, visando orientar a busca em espaços de solução complexos, simulando usos inteligentes de memória, com o objetivo de cruzar fronteiras de factibilidade ou otimalidade local. Suas regras, por serem gerais, são utilizadas como guias em outros métodos heurísticos. A cada iteração este algoritmo atualiza uma lista de soluções já visitadas de tal forma que apenas novas alternativas sejam testadas, aumentando a eficiência computacional.

Com base nas características dos métodos estudados escolhemos para o trabalho a heurística de Clarke & Wright, por ser a mais amplamente utilizada e também os Algoritmos Genéticos por serem particularmente adaptáveis à computação paralela. Em seguida serão apresentados com maiores detalhes alguns destes.

2.2.3. Modelo de Fisher e Jaikumar

Descreveremos a seguir um método tradicional para a resolução deste tipo de problema. Trata-se do modelo desenvolvido por Fisher e Jaikumar (1981) em sua publicação “A Generalized Assignment Heuristics for Vehicle Routing”. Sua compreensão facilitará o entendimento do problema e dos demais métodos que utilizaremos neste trabalho.

Através deste modelo, pode-se formular um algoritmo que forneça uma solução exata para um problema de roteamento de veículos. Este modelo também pode vir acompanhado de técnicas de relaxações quando o problema a ser tratado for mais complexo.

A seguir, apresentaremos todos os passos que compõem o modelo descrito, incluindo a sua formulação básica, que serve como base a diversos métodos de solução. Primeiramente, faz-se necessário descrever algumas condições para roteirização definidas pelo modelo:

- Nenhum cliente deve deixar de ser atendido;
- Todos os veículos iniciam e terminam seu trajeto no mesmo ponto;
- Um cliente deve ser atendido por apenas 1 veículo;
- A soma dos custos dos percursos deve ser minimizada.
- A capacidade do veículo deve ser respeitada

Formulação

Será definida agora a formulação do modelo de Fisher e Jaikumar aplicada a um problema de roteamento de veículos:

Índices:

i, j (1...N) - Endereço do cliente. Local de origem e destino de um percurso;

k (1...M) - Veículo que realizará o percurso.

Parâmetros

M - Número total de clientes de coleta em um dia;

N - Número total de veículos disponíveis;

C_{ij} - Custo de percorrer o percurso *i* ao *j*. No caso estudado, este custo está relacionado à distância a ser percorrida de *i* a *j*;

Q_k - Capacidade máxima do veículo *K* (peso ou volume);

q_i - É a demanda do cliente *i*.

Função Objetivo:

$$\min z = \sum_{i,j} \left(c_{ij} \cdot \sum_{k} X_{ijk} \right)$$

Sujeito a:

$$1 - \sum_{k=1}^{M} Y_{ik} = 1, \quad i = 2,..., N$$

Esta restrição garante que cada ponto (cliente) seja visitado por apenas um veículo, tendo em vista que este não necessitaria de mais de um veículo para coletar suas cargas.

$$2 - \sum_{k=1}^{M} Y_{ik} = M, \quad i = 1$$

Esta restrição garante que o ponto de partida (*i* = 1) receba a visita de todos os veículos. Ou seja, todos os arcos formados devem passar pelo ponto *i* = 1. Esta condição é verdadeira, já que todos os veículos devem retornar a empresa após realizarem suas coletas, de modo a descarregarem suas cargas.

$$3 - \sum_{i=1}^{N} q_i \cdot Y_{ik} \leq Q_k, \quad k = 1,..., M$$

Esta restrição assegura que a quantidade coletada não ultrapasse a capacidade do veículo. Ou seja, limita a utilização de cada veículo até sua capacidade máxima. Esta restrição é aplicável para o caso estudado.

$$4 - \sum_{j=1}^{N} X_{ij} = \sum_{j=1}^{N} X_{ji} = Y_{ik}, \quad i = 1,..., N \quad k = 1,..., M$$

Esta restrição garante que os veículos não interrompam suas rotas em um cliente. Ela relaciona as variáveis binárias *X* e *Y*, de modo que se um cliente *i*

for visitado por um veículo *k* (Y_{ik} = 1), haverá apenas um arco chegando e um arco saindo deste cliente. Caso contrário, todos os arcos receberão o valor nulo.

$$5 - \sum_{i,j \in S}^{N} X_{ijk} \leq |S| - 1, \quad \forall S \subseteq \{2,...,N\}, \quad k = 1,...,M$$

Esta restrição garante que não sejam formados arcos isolados ou "subrotas", ou seja, rotas fechadas isoladas que apesar de respeitarem as restrições anteriores não apresentem continuidade.

Este modelo é muito bom para entender a formulação básica do problema de roteamento de veículos, mas a sua aplicação prática se limita a problemas com poucos pontos a serem visitados e sem nenhum tipo de restrição operacional. Para resolver problemas maiores e mais complexos devemos utilizar outros procedimentos, tais como heurísticas construtivas e meta-heurísticas, que examinaremos no restante deste capítulo.

2.2.4. A Heurística de Clarke & Wright

Alguns problemas de roteirização de veículos são extremamente complexos, de modo que a solução ótima é quase impossível de ser encontrada. Para tais problemas, existem muitos modelos heurísticos que conseguem chegar a uma solução não exatamente ótima, mas aproximada do problema. Dentre estes, destaca-se o Algoritmo de Clarke & Wright, um modelo heurístico do tipo *saving* (economia) que busca substituir arcos mais caros dentro da rota por arcos de menor custo. O método a ser apresentado nesta seção é uma adaptação do algoritmo de Clarke & Wright (1962) publicada em "*Scheduling of Vehicles From a Central Depot to a Number of Delivery Points*", "*Operations Reseach*".

Este modelo possibilita a inclusão de restrições de janelas de tempo e

capacidades dos veículos, presentes no problema. Segundo Ballou (2001), a utilização deste algoritmo em problemas com um número limitado de restrições pode resultar em soluções próximas a 2% em relação à solução ótima. Por fim, o algoritmo de C&W é uma das técnicas mais conhecidas e utilizadas na resolução deste tipo de problema. Além de ser capaz de gerar soluções muito boas, ele é flexível para lidar com restrições, e relativamente rápido para problemas com um número moderado de paradas (Ballou, 2001).

Assumindo a existência de *N* pontos a serem visitados (lojas), partindo o veículo do depósito *0* e retornando ao mesmo após um ciclo. De momento, vamos admitir que uma solução inicial (a pior) seria a existência de *N* veículos disponíveis para realizar estas viagens. Cada veículo viaja do armazém até um cliente e retorna no fim do expediente. A mostra esta relação para 3 nós (2 clientes), sendo o nó *0* representando o Centro de Distribuição e os nós *i* e *j* os pontos de entrega. A distância total percorrida pelos dois veículos é:

$$D = 2 \cdot d_{0i} + 2 \cdot d_{0j}$$

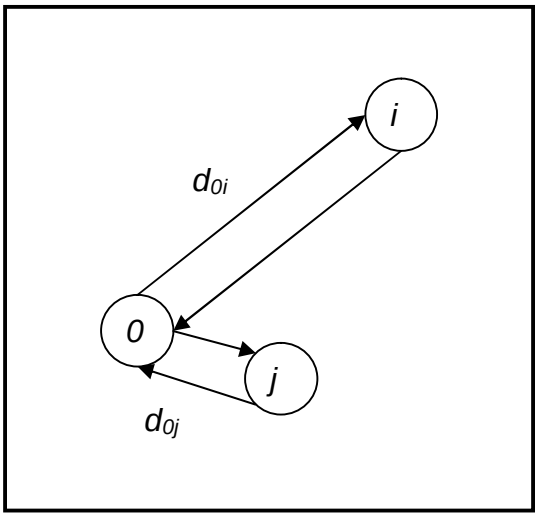

Figura 2.5 - Configuração inicial: uma rota para cada ponto (Elaborado Pelo Autor)

Agora, imaginemos que seja possível eliminar um veículo do roteiro acima, de forma que apenas um veículo percorra os três nós do problema. Assim, fazendo

o veículo percorrer o trecho *0 – i – j – 0*, há uma economia de distância percorrida, pois ele deixa de viajar um trecho *i – 0* e um trecho *0 – j*. No entanto, ele deve percorrer um trecho a mais *i – j*. Desta forma, a economia gerada por este novo percurso é representada por:

$$S_{ij} = d_{0i} + d_{0j} - d_{ij}$$

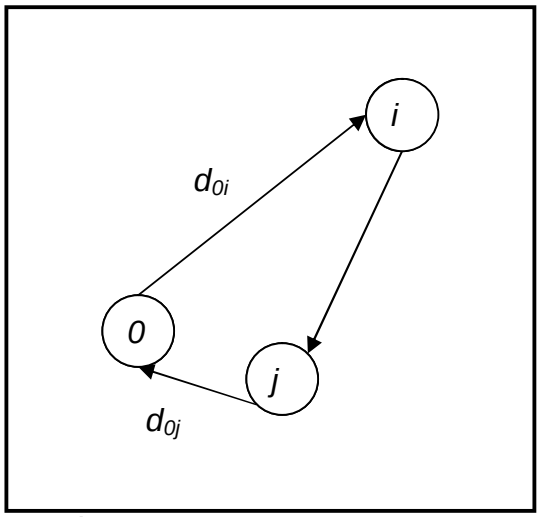

Figura 2.6 - Configuração após a junção dos pontos numa mesma rota (Elaborado Pelo Autor)

O método de Clarke & Wright baseia-se na computação destas economias. As economias representam o quanto a distância ou o custo podem ser reduzidos, agrupando nós (*i* e *j* no exemplo) e criando a rota *0 – i – j – 0*, que pode ser destinada a um veículo.

Para uma rede de *N* nós, computam-se as economias para cada par de nós, ordenam-se as economias obtidas em ordem decrescente, e constrói-se um roteiro ligando estes pares até concluir todas rotas. A descrição completa dos passos da heurística de Clarke & Wright encontra-se a seguir.

Descrição do Modelo

Nesta seção, descreveremos passo a passo a formulação da heurística de Clarke & Wright. Veremos também que existem duas versões do algoritmo: uma paralela e uma seqüencial. Apresentaremos a diferença entre as duas versões e definiremos qual será a versão usada no problema. Definiremos a seguir as restrições básicas do problema, tendo em mente que pode haver alterações dependendo da peculiaridade do problema tratado, por exemplo, a restrição de janelas de tempo. As restrições básicas do problema são:

- Cada rota inicia e termina no depósito;
- Cada cliente pertence somente a uma rota;
- A demanda de cada cliente não pode exceder a capacidade do veículo;
- A demanda de todos os clientes de uma rota não pode exceder a capacidade do veículo;
- O tempo total de um roteiro não excede a disponibilidade de tempo total de jornada de trabalho do motorista.

Objetivo: Atender todos os clientes, minimizando a distância total percorrida e respeitando todas as restrições impostas pelo problema.

A descrição da heurística segue, passo a passo, a seguir:

1 – Estabelecer como solução inicial para *N* nós, a formação de *N* rotas partindo e chegando ao depósito (esta solução, apesar de sempre factível, é a mais custosa).

2 – Computar as economias, S_{ij} ligando todos os nós *i* e *j*. Para *i* e *j* = nós 2,3,...,*N*. Onde *c* representa o custo de percorrer o trecho *i – j*, podendo ser a distância também; e *0* representa o depósito.

3 – Ordenar as economias obtidas em ordem decrescente, formando uma listagem de pares de nós, com suas respectivas economias.

Neste momento, o modelo de Clarke & Wright pode seguir dois caminhos:

1. *Versão Paralela*: efetua a melhor união factível;
2. *Versão Seqüencial*: realiza a extensão máxima de uma rota.

Versão Paralela

4 – Vá para o topo da lista de economias.

5 – Se, ligando os pares o resultado for uma rota factível de acordo com as restrições do problema, adicione esta ligação para a solução; caso contrário, elimine-a.

6 – Se ainda houver economias na lista, pule para a próxima e volte ao Passo 5. Se não houver mais economias, vá para o Passo 7.

7 – Fim.

Versão Seqüencial

4 – Vá para o topo da lista de economias.

5 – Se ligando os pares de nós *i* e *j* resulta numa rota factível, de acordo com as restrições do problema, faça esta união.

6 – Defina como rota atual a rota que contém os nós *i* e *j* unidos no Passo 5.

7 – Identifique os pontos extremos da rota atual e armazene-os nas variáveis *k* e *l*. (Por exemplo, após o Passo 5, $k = i$ e $l = j$)

8 – Determine a primeira economia S_{ik}, S_{kj}, S_{lj} ou S_{il} da lista que pode ser

utilizada para estender a rota atual. Note que a rota a ser unida à rota atual deve necessariamente começar com (0, *k*) ou (0, *l*), ou terminar com (*k*, 0) ou (*l*, 0). Se for impossível estender a rota atual vá para o Passo 10.

9 – Faça a união dos nós identificados e volte ao Passo 7.

10 – Volte para o topo da lista de economias e encontre a primeira economia que gere uma união factível. Se nenhuma economia for encontrada vá ao Passo 12.

11 – Faça a união dos nós identificados e defina como rota atual a rota que contém esses nós. Volte ao Passo 7.

12 - Fim.

A versão paralela da heurística de Clarke & Wright, por utilizar sempre os arcos de que proporcionam as maiores economias, gera na grande maioria das vezes soluções de menor custo que a versão seqüencial. Isto se deve muito pelo fato da versão seqüencial trabalhar uma rota por vez, tentando esticá-la ao máximo, fazendo que, com isso, use ligações de nós do final da lista, que geram menores economias. Por outro lado, esta característica da versão seqüencial faz com que ela priorize mais o aproveitamento de cada rota, utilizando em sua solução final, às vezes, menos rotas do que a versão paralela.

Ambas as versões da heurística podem ser utilizadas cabendo a escolha entre uma versão ou outra ao usuário. A eficiência da versão adotada varia de acordo com o tipo de problema. Neste trabalho iremos utilizar a versão paralela devido à produção de soluções de qualidade superior na maioria dos casos.

2.2.5. Algoritmos Meta-RaPS

Segundo Moraga et al., incorporar elementos aleatórios em uma heurística pode aumentar seu desempenho. Este procedimento, além de possuir baixo tempo de execução quando comparado a outras meta-heurísticas, é simples podendo ser utilizado em problemas práticos de roteamento de veículos.

O Meta-RaPS integra regras de prioridade (ou regras heurísticas), elementos aleatórios e amostragem. A cada iteração, este procedimento constrói soluções viáveis através da utilização de regras de prioridade em uma ordem aleatória. Após um número de iterações, a melhor solução encontrada é selecionada. A aplicação do Meta-RaPS em qualquer problema, como um procedimento geral, consiste em:

1 – Estudar a estrutura do problema e compreender de forma clara as variáveis, restrições e a otimização necessárias para resolver o problema;

2 – Encontrar regras heurísticas apropriadas à construção de soluções viáveis;

3 – Criar uma lista de próximas atividades disponíveis. Usando a heurística construtiva e a estrutura do problema, selecionar a próxima atividade. Um parâmetro percentual de restrição pode ser usado como um mecanismo para diminuir ou expandir a lista de atividades disponíveis;

4 – Modificar a regra de priorização introduzindo aleatoriedade de acordo com dois critérios: em alguns ou todos os passos da regra de priorização se possível; na combinação de regras de priorização;

5 – Rodar a heurística. A cada iteração, a regra de prioridade e sua versão modificada são combinadas produzindo diferentes soluções viáveis. Após um número de iterações a melhor solução encontrada é mantida

6 – No final de cada iteração o procedimento pode adotar uma técnica de melhoria na solução encontrada;

7 – O Meta-RaPS pode utilizar mecanismos para interromper a busca. O critério mais simples usado é estabelecer um número fixo de iterações depois do qual a melhor solução encontrada é mantida. Entretanto, pode ser implementado um critério de parada no caso de nenhuma solução melhor sem encontrada depois de um número de iterações ou parar quando o erro entre as duas melhores soluções encontradas atingir certo valor predeterminado.

Este procedimento bastante simples permite que uma série de soluções ligeiramente diferentes daquela gerada pela heurística construtiva original sejam testadas de maneira bastante rápida. Assim, ampliamos significativamente o universo de soluções exploradas permitindo que resultados melhores sejam obtidos.

Moraga et al. propõem três parâmetros básicos para controlar o funcionamento do Meta-RaPS:

Prioridade – é um fator que varia de 0 a 1 e representa a probabilidade da regra heurística original ser utilizada. Uma prioridade de 0,25 significa que 25% dos arcos testados para a formação de um roteiro serão escolhidos a partir da regra heurística original, que no caso do Clarke & Wright é a lista de economias. O demais 75% serão escolhidos aleatoriamente entre a lista restrita criada pelo algoritmo.

Restrição – este parâmetro controla o tamanho da lista restrita na qual o algoritmo buscará o próximo arco a ser incluído no roteiro. Um valor de 5% para este parâmetro significa que a lista restrita contará com todos os arcos cuja economia seja até 5% menor do que aquela proposta pela regra heurística (que representa a maior economia possível)

Melhoria –determina a probabilidade de um procedimento de melhoria ser utilizado no final de cada iteração. Assim, se este valor for de 20%, a cada 5 soluções geradas, uma será modificada através de melhorias.

2.2.6. Algoritmos Genéticos

Apesar de fornecer boas soluções para problemas de roteamento simples, quando há maiores restrições, as heurísticas construtivas não chegam a soluções tão boas. Por isso existem as meta-heurísticas que a cada iteração buscam melhorar a solução obtida na iteração anterior. Uma dessas meta-heurísticas utilizada na solução de problemas de roteamento de veículos e uma série de outras aplicações. São os Algoritmos Genéticos (AGs), que serão tratados nesta seção. Os AGs foram escolhidos como método para solução do problema tratado dentre as demais opções vistas no item 1 principalmente pela sua compatibilidade com a computação paralela como veremos a seguir.

2.2.6.1. Histórico

Nos anos 50, cientistas buscaram soluções para problemas evoluindo uma população de soluções possíveis através de conceitos da genética e da seleção natural. Já nos anos 60, Rechenberg criou um método que melhorava parâmetros reais de mecanismos, chamado “estratégias de evolução”. Em seguida surge a “programação evolutiva”, onde soluções representadas como máquinas de estado-finito, eram evoluídas através da mutação, para escolher ao final a mais adequada. John Holland nos anos 60, baseado no mecanismo de adaptação da natureza, cria os Algoritmos Genéticos, baseado em um quadro teórico para controlar essas adaptações, utilizando, *crossover*, mutação e inversão e uma população de cromossomos.

2.2.6.2. Evolução e Terminologia

De acordo com Mitchell M. (1996), a evolução vem sendo usada para a resolução de diversos problemas e para a termologia na computação, pois a natureza fornece a resolução de problemas complexos que buscam respostas entre inúmeras soluções e condições ambientais diferentes. Em um comparativo com a biologia, as várias alternativas são seqüências genéticas e a solução, indivíduos adaptados. O processo de evolução traz soluções originais para problemas complexos, e a utilidade de cada indivíduo dependerá de sua adaptação.

Todos os seres vivos têm células que possuem cromossomos. Nos AGs estes significam a possível resposta a um problema em forma de uma seqüência de bits, podendo ser dividido em genes ou blocos com funcionalidades diferentes no DNA, que codificam um elemento da possível solução. A variedade de possíveis características são chamadas alelos, 0 ou 1.

2.2.6.3. Elementos Básicos de um Algoritmo Genético

O algoritmo genético tem como características principais, populações de cromossomos, seleção pela utilidade, crossover reproduzindo novos indivíduos, e mutação aleatória. A seqüência de bits caracteriza um cromossomo sendo que cada localização do cromossomo possui dois possíveis alelos, 0 ou 1. O AG processa em seqüência as populações de cromossomos, substituindo uma população pela outra. Existe uma função que dá a cada cromossomo um valor. A utilidade de um cromossomo depende de como ele consegue resolver o problema, sendo a melhor solução aquela que apresentar maior utilidade. Estes elementos serão discutidos em maior detalhe nos próximos itens.

Codificação – é preciso delimitar os parâmetros do problema em símbolos ou genes, cujo comprimento pode ser constante ou dinâmico, codificando uma das soluções possíveis dentro de um espaço de soluções. Esses símbolos, que podem ser números binários, formam uma seqüência de bits, ou letras e valores. Os Ags, geralmente, são binários e constantes, seus parâmetros estão no mesmo espaço de valores, e em genes de mesmo tamanho.

Funções de Utilidade –Os AGs são aplicados para otimizar funções, onde o objetivo é encontrar um conjunto de parâmetros que maximize, por exemplo, uma determinada função multivariada complexa. são utilizados também em casos não-numéricos, em aminoácidos para buscar a melhor estrutura de uma proteína dentre uma população de soluções. Em suma, um algoritmo genético busca seqüências adaptadas, de acordo com sua função utilidade.

Operadores – existem três tipos principais de operadores no algoritmo genético, seleção, crossover e mutação. A seleção elege indivíduos de uma população para se reproduzirem, o mais adaptado dentre o espaço de soluções possíveis será selecionado. O operador de crossover escolhe um local no código genético e troca as seqüências, misturando dois pais e criando um descendente, e trocando bits de uma seqüência em um cromossomo aleatoriamente, a mutação, ocorre de acordo com uma pequena probabilidade, por exemplo, 0,01%.

2.2.6.4. Estrutura de um Algoritmo Genético Básico

Ao definir um problema com uma seqüência de bits representando as soluções possíveis, um algoritmo genético básico funcionaria da seguinte maneira:

1 – Gerar uma população aleatória com cromossomos de tamanho n;

2 – Nesta população para cada cromossomo *x*, calcular a função de utilidade *f(x)*;
3 – para criar n descendentes repetir as seguintes etapas:
- Na população atual, escolher os cromossomos pais de melhor utilidade.
- Em um ponto aleatório, cruzar a seqüência de bits dos pais, criando descendentes, dada uma probabilidade P_C ou taxa de crossover. Caso não ocorra crossover, os descendentes serão idênticos aos pais.
- Em alguns casos, os AGs usam uma taxa igual ao numero de pontos onde acontece cruzamento assim há possibilidade de crossover em pontos múltiplos, não em apenas um ponto.
- Com os dois descendentes fazer mutações com probabilidade P_M (probabilidade de mutação) em cada ponto das seqüências geradas.
- Colocar esses descendentes criados na nova população.
4 - Adotar essa nova população no lugar da antiga.
5 - Repetir a segunda etapa.

Cada iteração representa uma geração, sendo que ao termino da rodada, encontram-se indivíduos com maior utilidade, adaptados nesta população. Levando-se em conta que a grande aleatoriedade criara comportamentos diferenciados na população, para a resolução do mesmo problema, é usual, colher resultados estatísticos de muitas rodadas diferentes. Além desse procedimento existem detalhes como o tamanho da população e a escolha das taxas de *crossover* e mutação. Escolher corretamente esses parâmetros levará a um maior sucesso do algoritmo na resolução do problema.

2.2.6.5. Algoritmos Genéticos vs. Métodos de Busca Tradicionais

Na ciência da computação o termo busca tem diferentes significados, usamos esse termo até agora para descrever o que os AGs fazem, porém existem três sentidos para a palavra busca. Na Busca de Caminhos o problema é definir as

ações que a partir de um estado inicial levam eficientemente ao objetivo, já na Busca por Dados**,** o problema é achar informações que estão na memória. Finalmente há a Busca de soluções que visa encontrar dentre soluções possíveis a melhor solução. Neste tipo de busca é que será utilizado o algoritmo genético.

Existem outros métodos que também solucionam problemas do tipo “Busca de Soluções”, como simulated annealing, busca tabu e hill climbing. De um modo geral, as técnicas de “Busca de soluções” funcionam do seguinte modo: (1) geração de soluções candidatas; (2) avaliação destas por sua utilidade; (3) escolha das melhores; (4) usando um operador, novas variantes são produzidas sobre o conjunto restante.

A técnica de algoritmo genético difere das outras, pois combina elementos como busca maciçamente paralela, seleção com mutação e crossover. Esses elementos podem aparecer em outras técnicas isoladamente mais não nesta combinação completa, por isso parece ser um método bastante adequado para se resolver o problema de roteirização do Grupo varejista proposto neste trabalho.

2.3. COMPUTAÇÃO PARALELA

A computação paralela é uma técnica utilizada para proporcionar alta capacidade de processamento e desempenho computacional através do uso de múltiplos processadores trabalhando na mesma tarefa. Existem diversos níveis de paralelização, cada um deles correspondendo a uma infra-estrutura de hardware própria com características de desempenho e custos específicos. Além disso, o software utilizado na computação paralela deve ser adaptado para levar em conta a distribuição de tarefas entre os vários processadores. Nesta seção, discutiremos alguns dos principais aspectos desta abordagem

bem como o seu potencial para ser utilizada na solução do problema tratado.

2.3.1. Introdução e Conceitos Básicos

Em 1946, surgiu o primeiro computador digital eletrônico, que estabeleceu os conceitos básicos da computação. Em seguido o modelo de Von Neumann formado por processador, memória e dispositivos de E/S que executa seqüencialmente cada ordem dada pela unidade de controle. A partir daí, novas tecnologias foram criadas, como a computação paralela e as redes de computadores. O processamento paralelo faz a divisão de uma aplicação para vários elementos através de comunicação e sincronismo o que aumenta o desempenho do sistema. A computação paralela possui alguns conceitos básicos cuja compreensão nos ajudará no decorrer deste trabalho, os quais serão descritos a seguir.

Paralelismo e Concorrência – ocorre concorrência quando dois ou mais processos têm inicio no mesmo instante e não concluem suas atividades, o que pode acontecer tanto em um sistema com um processador quanto com múltiplos. Se um processo ocorre em paralelo, ele estará usando mais de um processador no mesmo intervalo de tempo. Dentro da computação existem três estilos de programação, a seqüencial que realiza uma tarefa depois da outra, a concorrente que inicia várias tarefas mesmo que a anterior não tenha acabado, e a paralela que inicia e executa as tarefas ao mesmo tempo.

Granulação – Para definir o tipo de plataforma, porte e quantidades de processadores onde se aplicará o paralelismo é necessário saber o tamanho das unidades de trabalho submetidas aos processadores, ou seja, o nível de granulação, que pode ser fina, media e grossa. A Granulação Fina usa o paralelismo para operações com processadores pequenos e simples, em grande quantidade. A Granulação Média situa-se entre a fina e a grossa. Já a

Granulação Grossa utiliza o paralelismo em processos e programas, com processadores grandes e complexos em pequena quantidade.

S*peedup* e Eficiência - Através do paralelismo, o processamento ganha velocidade, sendo assim, para verificar a qualidade dos algoritmos paralelos há duas medidas importantes, speedup e eficiência. O speedup é o aumento de velocidade ocorrida comparando-se a velocidade de *p* processadores com a de um na resolução do mesmo problema. O uso de uma granulação inadequada, ou partes do código sendo seqüenciais, causam uma sobrecarga na comunicação entre processadores e conseqüentemente uma diminuição no valor de speedup ideal, que deveria tender a *p*. A eficiência é a medida que trata dessa relação entre o speedup e o número de processadores. Quanto mais processadores são utilizados em paralelo, menor será a eficiência de processamento. No caso ideal (*speedup* = p), a eficiência seria máxima e teria valor 1 (100%).

2.3.2. Níveis de Paralelização

A paralelização pode ocorrer basicamente em três níveis distintos:

- Estação com múltiplos processadores: um computador conta com vários processadores integrados na sua arquitetura, com compartilhamento de memória e barramento único. É a forma mais cara de computação paralela, porém, tem operação fácil e melhor desempenho em tarefas seqüenciais.

- Cluster: sistema com dois ou mais computadores, cujo objetivo é fazer com que todo o processamento da aplicação e realização de tarefas seja distribuído aos computadores, de tal forma que ao usuário, pareça um único sistema respondendo. Pode ser montado com qualquer tipo de computador, conectados através de uma rede, o que permite que sejam montados a custos mais baixos,

além de ter escalabilidade, e alto controle de desempenho.

- Processamento Distribuído: permite maior paralelização, pois os computadores não precisam estar conectados o tempo todo e nem controlados de forma centralizada. O problema é quebrado em inúmeros sub-problemas que são enviados para computadores independentes, processados e finalmente devolvidos para o ponto de origem que irá consolidá-los. Esta paralelização é a mais barata, não requer investimento em hardware e não tem limite de processadores para se trabalhar. No entanto, há demora na obtenção de resultados, e impossibilidade de comunicação e controle dos processos.

Para os fins deste trabalho, os clusters se mostram como a opção mais prática por apresentarem custos mais reduzidos do que os supercomputadores e permitirem um alto grau de controle sobre os processos. A seguir iremos detalhar melhor o funcionamento e os tipos existentes de clusters.

2.3.3. Clusters

O uso de clusters eleva a confiança, distribuição de carga e capacidade de processamento. Cada computador de um cluster é denominado nó. Todos os nós formam uma rede, permitindo acréscimo ou retirada de um nó, sem interromper o funcionamento. O sistema operacional usado nos computadores deve ser de um mesmo tipo, ou seja, ou somente Windows, ou Linux, ou BSD, etc. Isso porque existem particularidades em cada sistema operacional.

Clusters são usados quando os conteúdos são críticos ou quando os serviços têm que estar disponíveis o mais rápido possível. Os pesquisadores, organizações e empresas estão utilizando os clusters porque precisam incrementar sua escalabilidade, gerenciamento de recursos, disponibilidade ou processamento num nível super-computacional e a um custo razoável.

Tipos de Clusters

Segundo Pitanga, M (2004), podemos classificar os clusters em alguns tipos básicos que veremos a seguir.

- Cluster Beowulf:

Estes clusters são usados para computação cientifica ou análises financeiras, tarefas típicas, que exigem alto poder de processamento. O Beowulf foi fundamentado em 1994, pela NASA, para processar as informações espaciais. Desde então, empresas (como HP e IBM) e universidades (como a brasileira Unesp e USP) vêm construindo clusters deste tipo, com cada vez mais nós.

O Beowulf tem um sistema dividido em um nó controlador denominado front-end (ou nó mestre), cuja função é controlar o cluster, monitorando e distribuindo as tarefas, atua como servidor de arquivos e executa o elo entre os usuários e o cluster. Os demais nós são os clientes, backends, ou nós escravos, e processam as tarefas enviadas pelo nó mestre. Nestes nós não existe a necessidade de teclados e monitores, e eventualmente até de discos rígidos (boot remoto), alem de poderem ser acessados via login remoto (telnet ou ssh).

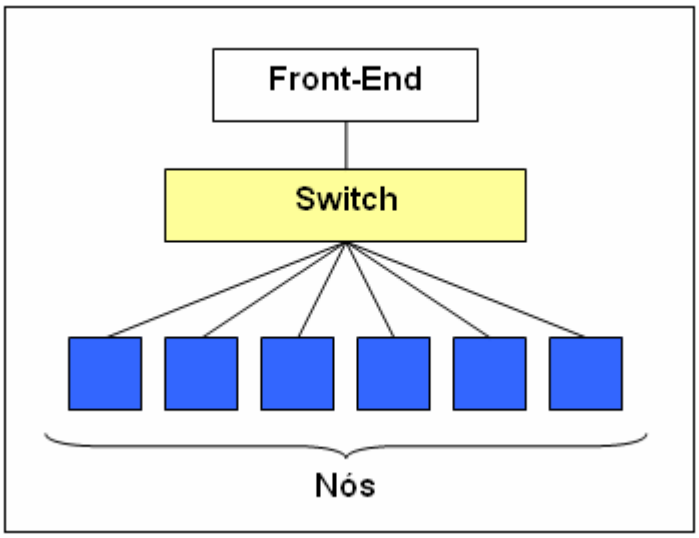

Figura 2.7 - Esquema simples de um Cluster Beowulf (elaborado pelo autor)

Este Cluster permite a construção de sistemas de processamento que podem alcançar altos valores de gigaflops (um gigaflops equivale a 1 bilhão de instruções de ponto flutuante executadas por segundo). Isso com o uso de computadores comuns e de um sistema operacional com código-fonte livre, além de equipamentos comuns às redes.

- Cluster para Alta Disponibilidade (High Availability - HA) – Para que o sistema não desative este modelo de cluster possui proteção e detecção de falhas, replicando serviços e servidores, através da redundância de hardware e reconfiguração de software em diferentes nós. Caso haja falha em um nó, aplicações e serviços estarão disponíveis em outro.

- Cluster para Balanceamento de Carga (Load Balancing) – este cluster integra seus nós para que todas as requisições sejam distribuídas entre eles. Todos os nós são responsáveis por controlar os pedidos. Se um nó falhar, as requisições são redistribuídas entre os nós disponíveis no momento, baseados em um escalonador e um algoritmo próprio.

- Cluster Combinado, HA & Load Balancing – É a união das características dos clusters de Alta Disponibilidade e de Balanceamento, aumentando assim a disponibilidade e escalabilidade de serviços e recursos, evitando paradas criticas. Suas características são o redirecionamento dos pedidos dos nós que falham para os nós reservas, melhoria na qualidade serviço e disponibilização de uma arquitetura escalonável.

2.3.4. Biblioteca de Paralelização MPI

As bibliotecas de paralelização são a alma do desenvolvimento dos aplicativos para os clusters, pois são elas que possibilitam que um mesmo aplicativo utilize

os recursos computacionais de mais de uma máquina ao mesmo tempo, através dos protocolos de comunicação. O cerne das aplicações paralelas é a utilização de mais de um processador para rodar a mesma aplicação, sendo que para se obter isso, é necessário que esses processadores se comuniquem, o que ocorre basicamente de duas formas: através de memória compartilhada ou da troca de mensagens.

A memória compartilhada é utilizada quando há mais de um processador em uma máquina. Para os usuários de máquinas pessoais e a maioria das empresas, não é comum existirem computadores com mais de dois processadores, devido ao seu alto custo. Dessa forma, esse tipo de paralelismo não é tão comum aos usuários. A memória distribuída é utilizada quando há mais de uma máquina trabalhando em conjunto, ou seja, cada máquina é independente, com processador e memória própria como no caso dos clusters. Por isso, é necessário que troquem informações entre si, o que é feito via rede.

As bibliotecas de paralelizações possuem um conjunto de instruções que permite que um computador se comunique com o outro, sendo os processos distribuídos e sincronizados. O MPI (Message Passing Interface) é uma das bibliotecas mais utilizadas hoje na programação paralela, e será adotada na construção dos algoritmos neste trabalho. Maiores detalhes sobre os tipos de biblioteca de paralelização e especificamente o MPI se encontram no Anexo A.

2.4. ALGORITIMO GENÉTICO PARALELO

Como o problema de roteamento de veículos com restrições operacionais é um problema de difícil solução, podemos utilizar as técnicas de computação paralela para aumentar a capacidade de processamento e assim chegar a soluções de melhor qualidade. Os algoritmos genéticos têm se mostrado bastante adequados para execução em máquinas paralelas devido ao alto grau

de paralelização intrínseco a sua estrutura. Existem dois tipos de algoritmos genéticos (AGs) paralelos que podem explorar os ganhos de escala desta abordagem de forma bastante eficiente: AGs mestre-escravo (ou global) e AGs de múltiplas populações (também conhecidos como modelo de ilhas).

A característica básica de um AG paralelo é a divisão da tarefa de avaliação das populações para os diferentes processadores disponíveis. A tarefa a ser paralelizada pode ser tanto a avaliação da função utilidade de cada indivíduo de uma população única, como a avaliação de uma sub-população inteira alocada a cada nó. Em ambos os casos existem alguns fatores importantes a serem avaliados para o bom desempenho do algoritmo.

Um fator crítico para qualquer AG é a troca de material genético de qualidade entre as soluções. Essa troca é influenciada tanto pela taxa de crossover quanto pela seleção de soluções que irão se reproduzir. Se a seleção for muito intensa a população irá convergir muito rápido e não haverá tempo suficiente para ocorrer uma mistura adequada entre os membros da população. Quando isso ocorre, o AG pode convergir para uma população sub-ótima. Por outro lado, se a seleção for muito baixa, o crossover pode destruir qualquer boa solução que tenha surgido e não tenha tido tempo de se reproduzir. Nesse caso o AG provavelmente não encontrará uma boa solução.

Além de decidir sobre os valores adequados para os parâmetros básicos do AG, na versão paralela é necessário escolher as taxas de migração, tamanho das sub-populações, topologia de conexões entre sub-populações e freqüência ou agendamento de migrações.

2.4.1. AGs Paralelos Mestre-Escravo

O mestre escravo é o tipo mais simples de AG paralelo. Essencialmente ele é

igual a um AG seqüencial que distribui o cálculo da função objetivo ou avaliação da utilidade para os nós disponíveis. O mestre armazena toda a população e cada nó escravo avalia uma fração dos indivíduos.

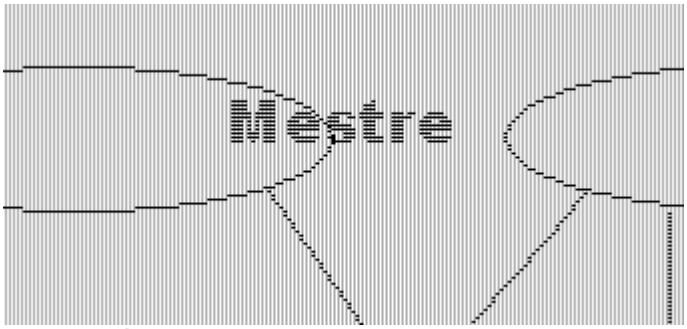

Figura 2.8 - Topologia básica de um AG Mestre-Escravo (adaptado de Cantú-Paz, 1999)

Segundo Cantú-Paz, os AGs mestre-escravo foram propostos por Grefenstette (1981), mas não tem sido largamente utilizados. No entanto, existem algumas aplicações bem sucedidas, como no caso de Fogarty e Huang (1991) onde os AGs são utilizados para evoluir regras para uma aplicação de balanceamento.

As operações de crossover e mutação podem ser paralelizadas. No entanto essas operações são tão simples que os ganhos obtidos com a paralelização podem ser perdidos com o aumento do tráfego de rede. O mesmo ocorre quando paralelizamos o processo de seleção de indivíduos.

Finalmente podemos dizer que o AG paralelo mestre-escravo ou global é um método de fácil implementação e pode ser um método eficiente quando requer cálculos demorados. Além disso, o método não altera a estratégia de busca e, portanto, podemos aplicar toda a teoria existentes para os AGs simples.

2.4.2. AGs Paralelos de Múltiplas Populações

O segundo tipo de AG que pode ser utilizado para aproveitar os ganhos da computação paralela é o de múltiplas populações. Eles também são chamados

de modelo de ilhas, e consistem de algumas sub-populações que freqüentemente trocam indivíduos segundo, Cantú-Paz (1999). Este é provavelmente o tipo mais popular de AG paralelo, mas ele é controlado por muitos parâmetros e a compreensão de como cada um deles afeta o desempenho do algoritmo em uma dada instância é muito trabalhosa.

Um dos aspectos mais importantes é o papel da troca de indivíduos entre as sub-populações. Esta troca é chamada de migração e é controlada por três parâmetros: (1) a taxa de migração, numero de indivíduos que vão migrar, (2) uma agenda de migração, que determina quando as migrações irão ocorrer, e (3) a tipologia de comunicação entre as sub-populações.

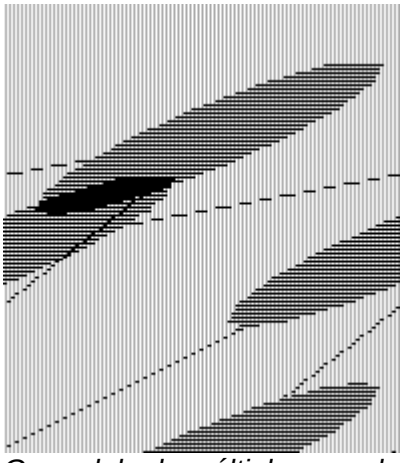

Figura 2.9 - Esquema de um AG paralelo de múltiplas populações. As sub-populações trocam indivíduos com seus vizinhos neste esquema (adaptado de Cantú-Paz, 1999)

As migrações afetam a qualidade da busca e a eficiência do algoritmo. Por exemplo, migrações freqüentes resultam em uma massiva troca de material genético, potencialmente útil, mas afetam negativamente o desempenho devido ao custo de comunicação. O mesmo ocorre com tipologias densamente conectadas, onde cada sub-população se comunica com muitas outras.

O objetivo final dos AGs paralelos é achar soluções rapidamente, sendo necessário, portanto, achar um balanço entre o custo de usar migração e o aumento das chances de se achar boas soluções.

3. MODELAGEM DO PROBLEMA

No capítulo 1 apresentamos a empresa e seus processos e descrevemos o problema que estamos nos propondo a resolver. No segundo capítulo foi feita uma ampla revisão da bibliografia sobre o tema, o que tornou possível a escolha de alguns métodos de solução que se mostraram mais apropriados ao problema, bem como a sua formulação.

Passaremos agora a combinar o conteúdo destes dois capítulos para construir um modelo capaz de proporcionar uma representação fiel das operações do grupo e ao mesmo tempo permitir que este possa ser resolvido para que sejam obtidas melhores soluções para o problema de abastecimento do que as utilizadas atualmente pela empresa.

Como foi visto no Item 1, o processo de abastecimento é composto basicamente por 3 etapas principais – recebimento dos pedidos, roteirização e transporte para as lojas. O foco deste trabalho, como já foi dito anteriormente, esta justamente na segunda etapa, a de roteirização. No entanto, para que esta possa ser resolvida a contento, são necessárias informações provenientes das duas etapas adjacentes. Na prática existe uma interação dinâmica entre as informações necessárias e disponíveis sobre um determinado processo e a forma como este pode ser modelado para sua resolução. Neste capítulo estaremos detalhando como o modelo foi concebido e no próximo será explicado como os dados necessários foram levantados e adaptados ao mesmo.

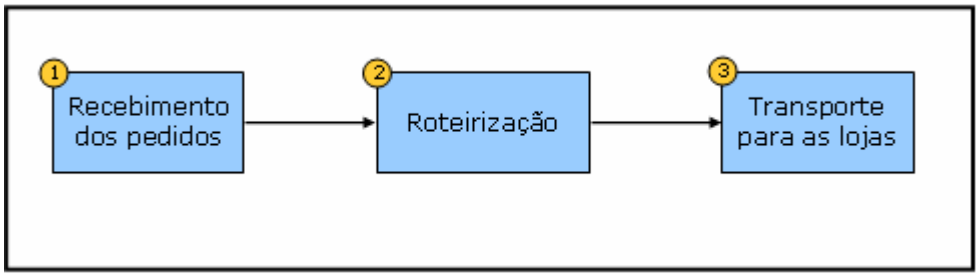

Figura 3.1 - Processo de abastecimento simplificado (elaborado pelo autor)

3.1. CLASSIFICAÇÃO DO PROBLEMA

O primeiro passo na construção do modelo é avaliar seus principais componentes e decidir como cada um deles será abordado. No Item 2.1.3 estudamos as possíveis variações dos problemas de roteirização de veículos segundo uma série de aspectos. De acordo com a classificação proposta por Assad (1998) podemos formular o problema da seguinte maneira:

Demanda

- O atendimento da demanda deve ser total, ou seja, nenhum dos clientes pode deixar de ser atendido e todo o volume solicitado deve ser entregue para cada um deles;
- Os produtos que compões os pedidos serão tratados como sendo iguais, considerando-se apenas o volume que ocupam e não o tipo de mercadoria;
- Não haverá prioridade de clientes de tal forma que o atendimento de todos eles seja igualmente importante;
- Serão consideradas apenas entregas, não havendo coletas ou backhaul;
- A demanda é previamente conhecida e não aleatória;

- Todas as entregas serão consideradas únicas e não periódicas de modo que a cada dia os pedidos devem ser formulados novamente.

Frota de Veículos

- A frota utilizada é heterogenia. Serão considerados no modelo os 3 tipos de veículos mais utilizados na prática, com custos e características diferentes;
    - Carreta – com capacidade de 28 paletes;
    - Truck – com capacidade de 14 paletes;
    - Leve – com capacidade de 7 paletes
- Os veículos possuem restrição de capacidade e todos eles devem partir do CD e retornar para o mesmo no final de cada roteiro;
- Não há restrições de carregamento dos veículos no CD, nem incompatibilidade entre veículos e carga (uma vez que não estamos considerando cargas líquidas ou refrigeradas no escopo deste trabalho);
- Há um número variável (ilimitado) de veículos que podem ser utilizados;
- Alguns veículos não podem visitar todas as lojas devido a restrições operacionais como altura das docas, falta de espaço para manobra e restrições de tráfego urbano.

Pessoal

- Será considerada uma jornada de trabalho máxima de 10 horas havendo a possibilidade desta ser ultrapassada apenas nos casos onde seja impossível atender uma determinada loja no tempo disponível;
- O número de motoristas será considerado ilimitado;
- O horário de início da jornada será flexível, de acordo com a necessidade de cada roteiro;
- Não haverá tempo extra de descanso entre as viagens (este será considerado no tempo de descarga que é superestimado).

Programação

- Serão consideradas janelas de recebimento rígidas para as entregas, ou seja, se o caminhão chegar antes do início da janela deverá aguardá-la e se chegar depois não poderá efetuar a entrega;
- O tempo de descarga será fixo em 1,5h;
- O centro de distribuição trabalha 24h por dia.

Informações

- Não estarão sendo utilizados dados geográficos e de redes viárias;
- As lojas serão localizadas de acordo com suas coordenadas de latitude e longitude (quando estas não estiverem disponíveis será utilizado o centróide da cidade onde as mesmas se situam);
- Os tempos de viajem serão calculados a partir da distância entre os pontos de origem e destino e da velocidade média dos caminhões naquele trecho (função da distância);

Os detalhes de como todas estas informações foram obtidas e processadas encontram-se no Capítulo 4.

3.2. PRINCIPAIS PARTICULARIDADES DO MODELO

Além das características discutidas na seção anterior, o modelo proposto neste trabalho apresenta duas diferenças importantes quando comparado aos modelos encontrados na bibliografia, que serão discutidas a seguir.

A primeira delas é a forma como o custo de transporte é considerado. Na grande maioria dos casos estudados, este custo é diretamente proporcional à distância percorrida pelos veículos durante o processo de entrega. No entanto, como no Grupo o abastecimento é feito por transportadoras terceirizadas, o custo irá depender dos contratos e não das distâncias. Basicamente, cada loja

está situada numa região para a qual existe um preço fixo de atendimento dependendo do modelo de veículo utilizado. Este preço independe do peso ou volume transportado e da distância ou tempo do percurso. A forma como esta característica particular foi incorporada aos métodos de solução será discutida em mais detalhes nos próximos capítulos.

Outra diferença significativa entre o modelo proposto e os encontrados na literatura é a possibilidade de que uma loja seja atendida por mais de um roteiro. Esta flexibilização do modelo se mostrou necessária uma vez que os volumes transportados para as lojas do Grupo são frequentemente superiores à capacidade dos caminhões.

Os modelos de roteirização tradicionais assumem que a demanda de cada ponto de entrega é menor do que a capacidade dos veículos. Caso eventualmente esta demanda seja superior, o problema é resolvido com o envio de caminhões totalmente ocupados exclusivamente para aquela loja até que a demanda remanescente seja inferior à capacidade dos caminhões.

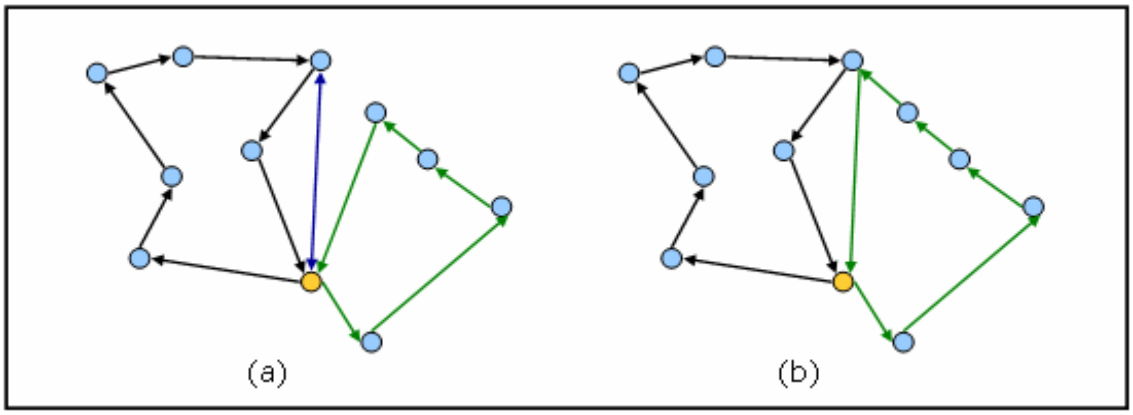

Figura 3.2 - (a) Apenas um roteiro pode passar por cada loja; (b) Possibilidade de dois roteiros passarem pela mesma loja (elaborado pelo autor)

A possibilidade de mais roteiros passando pela mesma loja amplia significativamente o número de soluções viáveis no espaço de soluções e torna mais trabalhosa a otimização dos roteiros. No entanto, esta flexibilidade

adicional permite que soluções mais vantajosas sejam testadas já que a capacidade ociosa dos caminhões em diferentes roteiros pode ser utilizada para atender a demanda de uma loja sem que haja a necessidade do envio de um veículo a mais para a mesma.

Mesmo assim, como serão testados diferentes métodos de solução para este modelo, em alguns casos tivemos que trabalhar com a restrição original devido à forma como o método trabalha.

No caso da heurística construtiva de Clarke and Wright, a proibição de múltiplas visitas a uma mesma loja foi utilizada tal como a literatura sugere, devido dificuldade de se adaptar o algoritmo para ignorar esta restrição, sem que sua qualidade seja prejudicada. Já para o Meta RaPS foi possível trabalhar com uma certa flexibilização desta restrição. Apesar de não permitir que múltiplos roteiros abasteçam uma mesma loja, o que pode ser implementado foi uma variação dos veículos que fazem a entrega exclusiva antes que a carga remanescente seja roteirizada. Desta forma, pudemos obter diferentes combinações de demanda que são atendidas na segunda fase do algoritmo obedecendo à regra de construção da heurística.

Finalmente, nos Algoritmos Genéticos, tanto na versão seqüencial quanto na paralela, a possibilidade de mais de um roteiro atender a mesma loja pode ser implementada graças a grande flexibilidade que este método proporciona na modelagem e tratamento de restrições.

No próximo capítulo estaremos descrevendo como as informações necessárias a resolução deste modelo foram obtidas e adaptadas.

4. LEVANTAMENTO DE DADOS

Uma etapa crítica na solução de qualquer problema consiste na obtenção e tratamento dos dados necessários à sua formulação. Na prática, a solução obtida por um modelo nunca será melhor do que a qualidade dos dados de entrada.

Neste capítulo iremos discutir os principais dados necessários à solução do problema de roteamento dos veículos que fazem o abastecimento das lojas da empresa, como os mesmos foram obtidos, tratados e inseridos nos modelos.

4.1. DISTÂNCIAS

Para se representar de maneira precisa a rede de lojas, de modo que possamos avaliar as distâncias percorridas nas rotas propostas, bem como o tempo necessário para percorrê-las, é necessário conhecer a localização de cada ponto.

As coordenadas de cada uma das lojas e do centro de distribuição em questão (latitude e longitude) foram obtidas a partir dos seus respectivos endereços com o auxílio de um site de localização (www.apontador.com.br). Para algumas lojas cujos endereços o software não conseguia encontrar, as coordenadas foram determinadas a partir do centróide da cidade. Estes dados já estavam disponíveis em uma planilha de cadastro das lojas e centros de distribuição, que além destas informações, contém o nome da loja, telefone, endereço completo, CNPJ, entre outras.

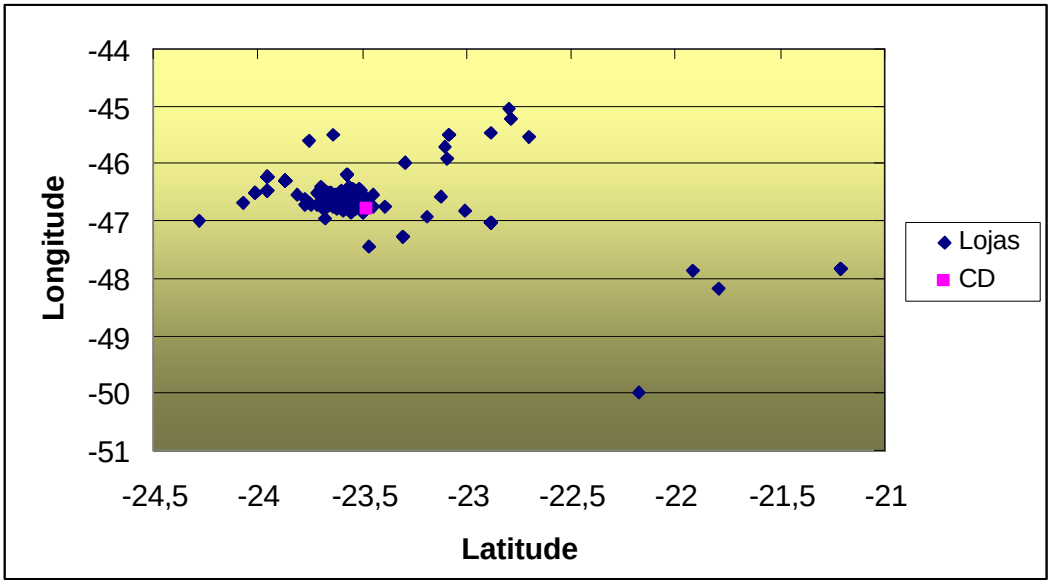

Figura 4.1 - Localização das lojas e CD do Grupo (elaborado pelo autor)

A Figura 4.1 apresenta a latitude e longitude levantada para cada uma das lojas incluídas no escopo do trabalho. Como podemos perceber, existe um grupo de lojas mais afastado que foi mantido no problema por ser atendido pelo CD em questão e estar localizado dentro do estado de São Paulo. Por estas lojas apresentarem uma distância mais elevada, o tempo de viagem também cresce bastante. Em alguns casos na prática este tempo ultrapassa a jornada de trabalho do motorista, o que não deveria ser permitido. Para lidar com este fato no modelo, em cada um dos métodos de solução propostos, serão feitas adaptações para permitir que as restrições de tempo máximo possam ser violadas caso seja impossível atender uma determinada loja neste intervalo, viabilizando a solução do problema.

De posse das coordenadas podemos calcular a distância entre todos os pares de pontos que compõem a rede. Caso estas coordenadas fossem a posição (x, y) dos pontos sobre um plano poderíamos obter as distâncias calculando apenas o comprimento da reta que os une como sendo:

$$d_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$

No entanto, como as coordenadas estão em latitude e longitude, devemos determinar o comprimento do arco que une os dois pontos sobre a superfície da Terra. Este comprimento pode ser obtido através da seguinte equação:

$$d_{ij} = 6377 \cdot a\cos\left(sen\left(lat_i\right) \cdot sen\left(lat_j\right) + \cos\left(lat_i\right) \cdot \cos\left(lat_j\right) \cdot \cos\left(abs\left(lon_j - lon_i\right)\right)\right)$$

Onde:

- 6377 é o raio da Terra em quilômetros;
- lat é a latitude de um ponto expressa em radianos;
- lon é a longitude de um ponto expressa em radianos;
- sen é a função seno;
- cos é a função cosseno;
- acos é a função arco cosseno;
- abs é o valor absoluto de um número;

Uma vez determinadas as distâncias dos arcos que unem os pontos da rede dois-a-dois, temos uma boa aproximação da distância real a ser percorrida para ir de um destes pontos ao outro. No entanto esta distância real, na prática, irá depender das vias terrestres disponíveis entre os pontos.

Para corrigir esta distorção entre o valor teórico e o valor prático das distâncias foi feita uma análise com base nos dados históricos de distância percorrida e do resultado da equação anterior. As diferenças obtidas podem ser vistas no gráfico da Figura 4.2.

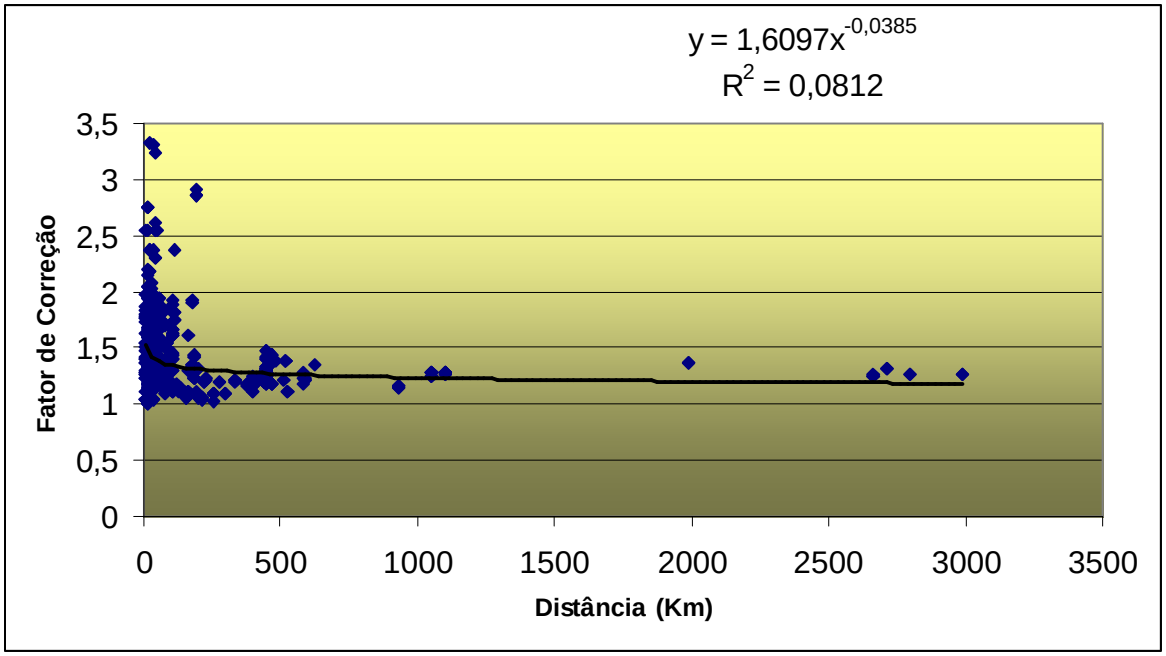

Figura 4.2 - Correlação entre o fator de correção e a distância (elaborado pelo autor)

Como podemos observar, para distâncias pequenas, a razão entre as duas distâncias apresenta uma variabilidade muito grande de acordo com o trajeto percorrido. Já para distâncias maiores este fator tende a um valor médio e sua variabilidade diminui. Com o intuito de representar este comportamento de uma maneira que tornasse mais realistas as distâncias efetivas que usaremos nos modelos de solução tentamos ajustar diversos tipos de curva que representassem esta distribuição. A curva que pode ser vista no gráfico foi a que melhor se ajustou aos pontos, mas mesmo assim de maneira muito pobre, apresentando um r^2 apenas 0,08. Desta formas optamos por utilizar um fator de correção fixo de 1,27 que representa o valor médio dos números analisados.

A maneira mais acurada de se fazer uma estimativa das distâncias seria utilizar os dados históricos de distância percorrida para cada combinação de pontos. No entanto, estes dados só existem para uma fração ínfima dos 23.005 (215 combinados dois a dois) arcos necessários para representar a rede. Outra maneira seria obter esta distância por meio de um software de cálculo da distância mínima entre dois pontos que utiliza um banco de dados de

informações geográficas incluindo mapas viários detalhados. Como não havia uma ferramenta deste tipo disponível e o número de arcos a serem calculados era muito grande optamos pelo uso do fator de correção mencionado.

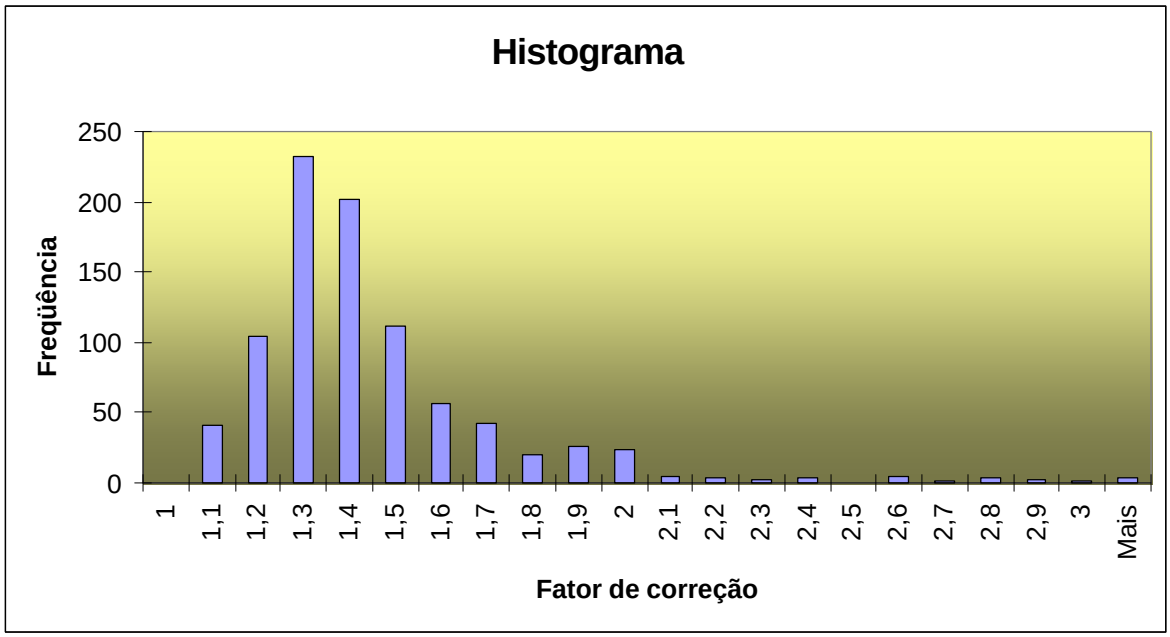

Figura 4.3 - Gráfico de distribuição dos fatores de correção (elaborado pelo autor)

4.2. VELOCIDADE DOS VEÍCULOS

De posse das distâncias de cada um dos trechos que poderão compor uma rota de entrega, devemos calcular o tempo que os respectivos trechos levarão para serem percorridos. Para tanto é necessário saber a velocidade média desenvolvida por um veículo ao percorrer um determinado trecho.

Com o intuito de se determinar esta velocidade, novamente fizemos uma análise dos dados históricos do grupo. Desta vez comparamos a distância real percorrida entre dois pontos e o tempo total de percurso (sem incluir as paradas para carga e descarga do veículo). Os resultados obtidos podem ser vistos na Figura 4.4 que conta com 1.000 observações.

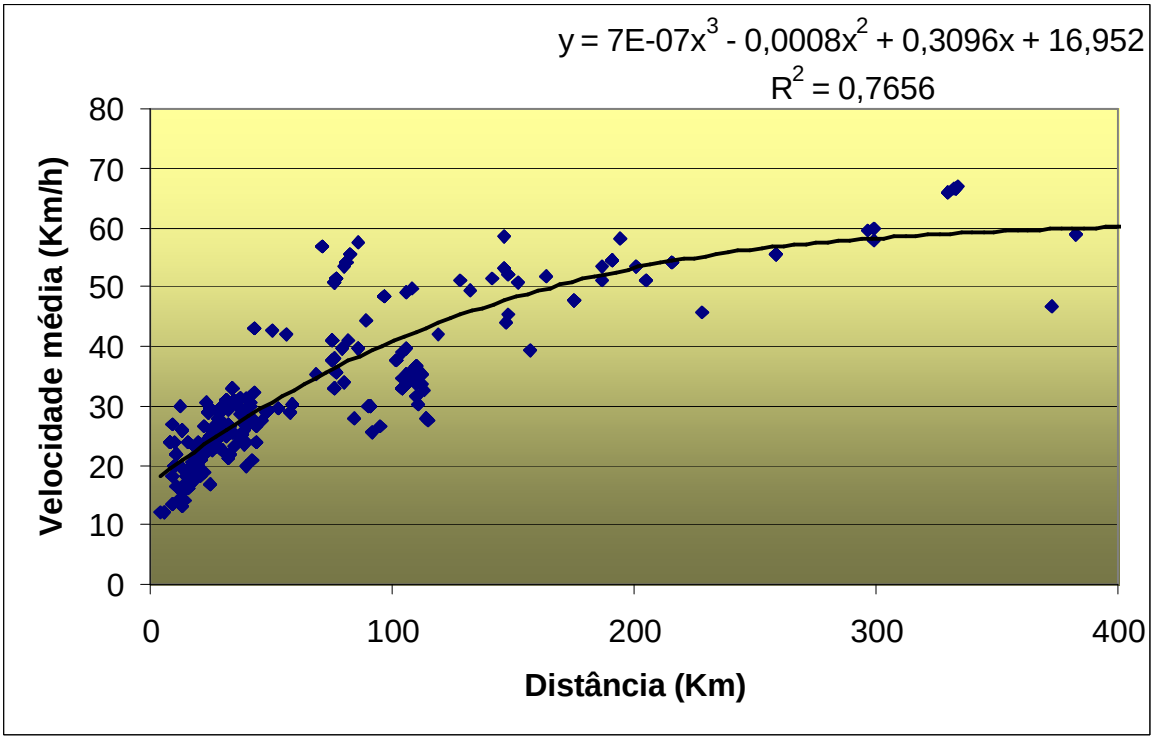

Figura 4.4 - Gráfico velocidade média desenvolvida de acordo com a distância do trecho (elaborado pelo autor)

Ao contrário do caso em que analisamos o comportamento da distância teórica e real, agora é nítida a existência de uma correlação. À medida que as viagens se tornam mais longas, a velocidade média tende a aumentar. Isto se deve ao fato de que, para se percorrer grandes distâncias utilizam-se, na maioria das vezes, estradas nas quais a velocidade desenvolvida é maior. Já nos trechos curtos, é comum o tráfego em vias urbanas onde há maiores congestionamentos e semáforos que reduzem a velocidade média.

Para representar este fato no modelo de solução tentamos ajustar algumas curvas que representassem bem este conjunto de pontos. Uma curva logarítmica parece se adequar bem aos dados analisados, mas ela tem o problema de tender a zero quando a distância é pequena. Assim, optamos por uma equação do terceiro grau que apresenta um r^2 de 0,766 e representa muito bem os pontos no intervalo que utilizaremos na prática. Para evitar que a

velocidade se torne muito alta para grandes distâncias já que o termo ao cubo é positivo, fixamos um limite em 60km/h que na prática é bastante razoável.

Desta forma, a velocidade é dada pela equação abaixo:

$$V = \min\left(0{,}0000007 \cdot d^3 - 0{,}0008 \cdot d^2 + 0{,}3096 \cdot d + 16{,}952 \quad , \quad 60\right)$$

4.3. FROTA DE VEÍCULOS

O próximo passo no levantamento dos dados necessários a resolução do problema é conhecer a frota de veículos disponíveis para realizar as entregas, assim como sua estrutura de custos.

O Grupo não possui nenhum veículo próprio para realizar o abastecimento de suas lojas. Todas as entregas são feitas através de empresas terceirizadas que disponibilizam caminhões dedicados ao atendimento das necessidades da operação. No total, são mais de 40 transportadoras que prestam serviços para a empresa, o que garante uma ampla variedade de veículos e disponibilidade. Cerca de 500 veículos compõe a frota dedicada, mas este número pode ser adaptado de acordo com as necessidades. A Tabela 4.1 lista os modelos de veículos utilizados nas operações da empresa:

Modelos disponíveis	
Carreta mono eixo com baú e plataforma	Toco frigorífico com plataforma
Carreta com baú sem plataforma	Toco frigorífico sem plataforma
Leve baú com plataforma	Truck baú com plataforma longo
Leve baú sem plataforma	Truck baú com plataforma
Leve frigorífico sem plataforma	Truck baú sem plataforma
Toco baú com plataforma	Truck frigorífico sem plataforma
Toco baú sem plataforma	Truck isotérmico com plataforma

Tabela 4.1 - Lista de veículos utilizados nas operações (elaborado pelo autor)

Apesar desta grande variedade de veículos disponíveis, na pratica, apenas alguns deles são utilizados nas operações de abastecimento de carga paletizada com a qual estaremos lidando ao longo deste trabalho. Os modelos mais importantes neste processo são as carretas, trucks e veículos leves, que são utilizados no transporte de produtos de mercearia, principal categoria considerada no escopo deste trabalho. Maiores detalhes destes veículos encontram-se na tabela a seguir:

Veículo	Capacidade (Kg)	Qtd. Paletes	Cubagem (m^3)	Medidas Internas / Externas (m)		
				Comprimento	Largura	Altura
Leve	9.000	7	19,7	4,79 / 5,00	2,06 / 2,20	2,00 / 2,24
Truck	22.000	14	41,2	7,29 / 7,50	2,46 / 2,60	2,30 / 2,56
Carreta	44.000	28	85,0	14,40 / 14,60	2,46 / 2,60	2,40 / 2,66

Tabela 4.2 - Detalhes dos veículos (elaborado pelo autor)

Tendo sido definidos os centros de distribuição, as lojas e os veículos que serão utilizados nos modelos de solução, se faz necessário conhecer os custos incorridos ao se fazer uma viagem de um ponto a outro nesta rede com um determinado veículo. Neste sentido dois pontos importantes devem ser esclarecidos.

Em primeiro lugar, como a frota é terceirizada, os custos com os quais estaremos trabalhando não são os custos reais da operação (combustível, depreciação dos veículos, salário dos motoristas, etc.), mas sim o valor estabelecido em contrato com as transportadoras.

O segundo ponto importante é que este preço pago por viagem não depende diretamente da distância total percorrida, tempo de percurso, ou mesmo do peso ou volume transportado. Este valor esta definido como um preço fixo pago por viagem de acordo com uma classificação de localidade onde se encontram as lojas.

De acordo com este modelo de custeio, duas lojas localizadas na região de Campinas, por exemplo, terão o mesmo custo de atendimento mesmo que uma delas esteja mais longe ou necessite de um volume maior. É evidente que este custo fixo é definido por tipo de veículo de tal forma que uma viagem de carreta custará necessariamente mais do que uma realizada por um veículo leve para uma mesma distância. Finalmente é importante notar que caso um veículo seja designado para atender mais de uma loja na mesma viajem, o preço cobrado será sobre a localidade mais cara, além de um adicional de R$16,00 por cada loja extra visitada.

Concluindo, podemos dizer que os fatores determinantes do custo de frete são:

- Tipo do veículo – quanto maior a capacidade mais caro o frete;
- Região onde a loja se encontra – quanto mais distante a região mais caro o frete, porém este é constante dentro de uma mesma região;
- Número de lojas no roteiro – custo extra de R$16,00 para cada loja além da primeira em cada viagem (independentemente da região ou veículo);
- Combinação de lojas no roteiro – caso existam lojas de duas regiões num mesmo itinerário o frete será cobrado pela mais cara.

Região	**Custo de Frete**		
	Carreta	**Truck**	**Leve**
Americana / Sta. Barbara	R$ 627,90	R$ 408,20	R$ 288,60
Araçatuba	R$ 2.642,90	R$ 1.733,94	R$ 1.235,26
Araraquara	R$ 1.362,40	R$ 889,20	R$ 630,50
Atibaia	R$ 383,50	R$ 256,10	R$ 184,60
Bauru	R$ 1.765,40	R$ 1.153,36	R$ 818,74
Botucatu	R$ 1.257,10	R$ 817,18	R$ 579,02
Caçapava	R$ 626,60	R$ 413,14	R$ 295,36
Campinas	R$ 509,60	R$ 331,76	R$ 235,04
Campos do Jordão	R$ 912,60	R$ 604,24	R$ 433,16

Tabela 4.3 - Exemplo de custos de frete por região e modelo de veículo (elaborado pelo autor)

4.4. DEMANDA

As lojas da empresa, trabalham com uma ampla variedade de produtos. Dentre as categorias mais importantes podemos citar frutas, legumes e verduras, mercearia, carnes e outros produtos refrigerados, eletrodomésticos, etc. Como já mencionamos na definição do problema, estaremos abordando neste trabalho o abastecimento das lojas com cargas paletizadas, que podem ser transportadas em caminhões comuns.

A demanda das lojas, é composta por diferentes categorias de carga paletizada, quais sejam contentores, paletes, gaiolas, roltainers e carrinhos etc. Todas estas categorias de carga podem ser tratadas de forma análoga, exceto pelos carrinhos e gaiolas que ocupam metade de uma posição palete no veículo. Assim, a demanda total de uma determinada loja é obtida pela consolidação de toda a carga demandada, respeitando-se o volume que ela ocupa.

Para o propósito deste trabalho, escolhemos algumas datas com uma demanda típica para serem utilizadas como problema a ser resolvido. A Tabela 4.4 a seguir representa a consolidação da demanda de algumas lojas numa das datas escolhidas. A demanda total neste dia foi de 2269 paletes, o que nos dá uma idéia do tamanho do problema com o qual estamos lidando.

Loja	Demanda por tipo de carga							
	VDE	PLT	AZU	GLA	PTO	ROL	CAR	TOTAL
1	0	13	2	2	0	0	0	16
2	0	29	1	2	0	0	0	31
3	0	4	0	0	0	0	0	4
4	0	5	0	0	0	0	0	5
5	0	5	0	0	0	0	0	5
6	0	5	0	1	0	0	0	5,5
7	0	0	0	0	0	20	0	20
8	0	7	0	2	0	0	0	8
…	…	…	…	…	…	…	…	…

Tabela 4.4 - Consolidação das cargas no dia escolhido (elaborado pelo autor)

Carga	Descrição
VDE	Contentores Verdes
PLT	Paletes PBR
AZU	Paletes Azuis
GLA	Gaiolas
PTO	Contentores Pretos
ROL	Rolltainers
CAR	Carrinhos

Tabela 4.5 - Descrição dos tipo de carga (elaborado pelo autor)

4.5. JANELAS DE RECEBIMENTO

Uma das restrições operacionais que torna este um problema complexo é a existência de janelas de recebimento muito diferentes entre as lojas. Esta janela pode ser definida por dois horários, um inicial (*hi*) e um final (*hf*) entre os quais as entregas podem ser efetuadas. Caso um veículo chegue em uma loja antes de sua janela de recebimento, o mesmo deverá ficar aguardando até o momento *hi*. Por outro lado, se ele chegar após *hf* a entrega não poderá ser efetuada e a rota é considerada inviável.

A Tabela 4.6 mostra as janelas de recebimento para algumas das lojas da rede. Repare que podem existir casos onde o horário final é menor do que o inicial. Isto representa que a loja pode receber mercadorias durante a madrugada de um dia para o outro. Para representar este fato no modelo, somamos 24 horas no horário final do recebimento quando necessário, fazendo com que a janelas ficassem com a duração correta. Os dados completos podem ser observados no Anexo C.

Loja	Distância (Km)	Início do Recebimento (h_i)	Final do Recebimento (h_f)
1	23	12:00	15:00
2	13	18:00	20:00
3	17	11:00	15:00
4	16	06:00	08:00
5	17	19:00	21:30
6	22	13:00	18:00
7	88	08:00	14:00
8	21	19:00	22:00
9	114	06:00	22:00
10	155	06:00	07:00
11	19	20:00	22:00
…	…	…	…

Tabela 4.6 - Janelas de recebimento (elaborado pelo autor)

O horário no qual um veículo parte do CD, que também consiste numa variável de decisão, é determinado com base na janela de recebimento da primeira loja da rota. Este horário será igual a *hi* desta loja menos o tempo de percurso do trecho do CD até a mesma. Desta forma, o veículo chegará à loja no momento mais cedo em que pode realizar a entrega.

O levantamento de dados foi fundamental para a resolução do problema. Este processo nos permitiu conhecer melhor as operações da empresa e nos deu subsídios para a construção dos modelos. No próximo capítulo, passaremos a descrever em detalhes todos os modelos desenvolvidos neste trabalho e os métodos utilizados em cada um deles.

5. RESOLUÇÃO DO MODELO

Após ter estudado os principais aspectos relacionados ao problema que nos propomos a resolver, feito uma ampla revisão bibliográfica sobre os tipos de problemas e os métodos de solução existente, e ter detalhado o modelo desenvolvido e os dados levantados, passaremos agora a descrever como estes foram utilizados no desenvolvimento dos métodos de solução. Este capítulo está dividido em quatro partes de acordo com os métodos utilizados.

Na primeira parte trataremos do método básico utilizando heurística de Clarke & Wright, e todas as modificações feitas para adaptá-lo às condições reais do problema, principalmente quanto ao cálculo de economias, a frota heterogênea e as lojas que não podem receber carretas. Na segunda parte falaremos do modelo que utiliza a meta heurística conhecida como Meta-RaPS, que é uma forma simples de se melhorar o desempenho de uma heurística construtiva.

Na terceira parte descreveremos o modelo que utiliza a meta-heurística de Algoritmos Genéticos na resolução do problema. Finalmente, na quarta parte será explicado como o Algoritmo Genético foi adaptado para funcionar de forma paralela em um cluster.

5.1. ALGORITMO ADAPTADO DE CLARKE AND WRIGHT

Nesta seção iremos detalhar como o algoritmo básico foi adaptado de forma a atender todas as restrições práticas do problema e os resultados obtidos pelo uso desta heurística.

5.1.1. Atendimento da Demanda

Como foi observado na revisão da literatura sobre o assunto, uma das hipóteses básicas da grande maioria dos métodos de solução, inclusive da heurística de Clarke & Wright, é que a demanda de cada uma das lojas não pode exceder a capacidade dos veículos, pois neste caso a mesma não poderia ser atendida em apenas uma parada. No entanto, conforme mencionado na Seção 3.2, a maioria das lojas possui demanda superior à comportada pelos veículos que podem atendê-las. Assim, corrigir esta distorção passa a ser, no problema prático que estamos resolvendo, uma questão fundamental.

Neste sentido, foi implementado um algoritmo para suprir a demanda de cada loja com caminhões lotados visitando exclusivamente aquela loja até que a demanda remanescente seja menor do que a capacidade do maior veículo capaz de atendê-la. Assim, garantimos que a demanda de cada loja que será roteirizada pela heurística é menor do que a capacidade dos caminhões, permitindo que os métodos pesquisados sejam utilizados sem maiores problemas. Na prática, este método é bastante intuitivo, pois enviar veículos grandes com carga completa é a forma de transporte que minimiza o custo unitário por palete entregue.

O algoritmo proposto segue os seguintes passos:

- Para cada loja
    - Verifique qual o maior veículo que pode atender a loja
    - Enquanto a demanda remanescente for maior do que a capacidade deste veículo
        - Envie um caminhão deste tipo com carga completa
        - Recalcule a demanda remanescente

5.1.2. Cálculo das Economias

Como vimos anteriormente, o objetivo da heurística é minimizar a distância total percorrida em todas as rotas, assim, a economia que desejamos computar é a distância economizada ao juntar duas lojas numa mesma rota. No entanto, na operação analisada, o custo de frete pago às empresas transportadoras não é diretamente proporcional à distância percorrida pelo caminhão. Este custo é calculado com base em regiões geográficas pré-definidas, e negociado a priori. Cada uma destas regiões possui um custo fixo por viagem que chamaremos de C_i, independentemente da distância real percorrida. Caso o caminhão seja designado para atender mais de uma loja na mesma viajem, é cobrado além de C_i um custo extra para cada outra loja atendida ($Delta_i$). Se as lojas estiverem em regiões cujos custos de atendimento sejam diferentes entre si, será cobrado pelo maior valor. Desta forma, o custo de atender uma determinada rota pode ser definido como:

$$CR = \max(C_1, C_2, ..., C_n) + (N-1) \cdot \max(Delta_1, Delta_2, ..., Delta_n)$$

Onde N representa o número de lojas atendidas pela rota e CR o custo total da rota.

Sabendo que o custo de frete é calculado deste modo, podemos adaptar o conceito original de economia proposto no algoritmo para adequá-lo ao caso prático. Vejamos o que ocorre no exemplo abaixo:

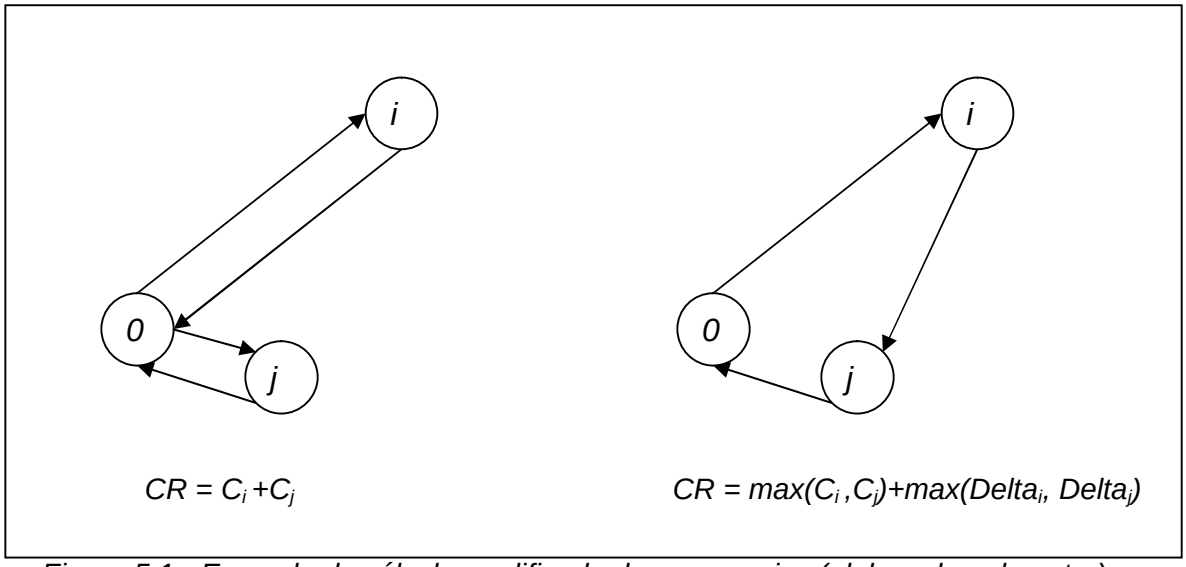

Figura 5.1 - Exemplo do cálculo modificado das economias (elaborado pelo autor)

Neste caso onde CR é o custo de uma rota, assumindo que $C_i > Cj$ e $Delta_i > Delta_j$, a economia obtida pela junção das lojas i e j na mesma rota seria:

$$S_{ij} = C_j - Delta_j$$

Num caso genérico teríamos:

$$S_{ij} = min(C_i, C_j) - min(Delta_i, Delta_j)$$

5.1.3. Frota Heterogênea

Ao contrario do que ocorre na heurística básica de Clarke & Wright, no modelo que estamos criando existe a possibilidade de se utilizar diversos tipos de veículos para percorrer cada rota. Este fato adiciona maior complexidade ao problema e deve ser cuidadosamente incorporado do algoritmo desenvolvido para sua solução. Para lidar com esse fato o algoritmo funciona da seguinte maneira:

- Assim como no algoritmo original, Inicialmente são criadas rotas individuais para atender cada uma das lojas;

- A cada rota é atribuído o menor caminhão capaz de atender a demanda da respectiva loja;

- Quando duas lojas são avaliadas para se juntar em uma mesma rota, caso a demanda das duas combinadas ultrapasse a capacidade do veiculo atual, o menor veiculo capaz de atender essa demanda é designado para a rota.

Seguindo estes passos durante a execução da heurística de Clarke & Wright, garantimos que no final do processo, apenas rotas que respeitem a capacidade dos veículos serão formadas e que os menores veículos possíveis serão alocados para cada rota. Desta forma, caso seja possível atender uma rota com um veículo menor, estaremos minimizando os gastos com fretes uma vez que os veículos menores possuem custo fixo mais baixo por viagem.

Por exemplo, considere duas lojas cujas demandas sejam de 5 paletes cada. Inicialmente, a cada uma delas será atribuída uma rota percorrida por um caminhão Leve cuja capacidade é de 7 paletes. Caso estas duas lojas sejam unificadas num mesmo roteiro, de acordo com a regra que acabamos de definir, a demanda combinada de 10 paletes passaria a ser atendida por um Truck com capacidade de 14 paletes. Supondo um custo fixo de frete igual a R$100,00 para o veículo Leve e R$140,00 para o Truck, o custo de atendimento destas duas lojas passaria de R$200,00 (2x R$100,00) para R$156,00 (R$140,00 + R$16,00 devido a loja extra na mesma rota), resultando numa economia de R$44,00.

5.1.4. Restrições no Recebimento

Com a utilização de diversos modelos de veículos, surge um problema que não ocorre quando a frota é homogenia. Trata-se da impossibilidade de algumas lojas receberem determinados tipos caminhões por suas dimensões não serem compatíveis com as docas de recebimento disponíveis ou não haver espaço de manobra, entre outros motivos. No caso do Grupo pudemos classificar as lojas em duas categorias, as que aceitam todos os tipos de veículos e as que não podem receber carretas.

Foram necessárias algumas adaptações na heurística original para que ela pudesse trabalhar com esta nova restrição, principalmente no atendimento inicial da demanda, como já foi explicado no início desta seção, e na avaliação da viabilidade na junção de duas lojas numa rota. Neste segundo ponto as mudanças implementadas foram as seguintes:

- Quando duas lojas são consideradas para a formação de uma rota:
    - Caso todas as demais restrições sejam respeitadas:
        - Se a demanda considerando a nova loja for menor do que a capacidade do segundo maior caminhão
            - Faça a junção das rotas.
        - Se esta demanda for maior do que a capacidade da carreta:
            - Não faça a junção.
        - Se a demanda ficar entre a capacidade da carreta e do segundo maior caminhão:
            - Se todas as lojas da rota incluindo a nova podem receber carretas:
                - Faça a junção das rotas.
            - Se pelo menos uma loja não aceita carreta:
                - Não faça a junção.

Deste modo garantimos que uma determinada rota formada só poderá ser atendida por carretas se cada uma das lojas que a compõe seja capaz de receber carretas.

5.2. ALGORITMO ADAPTADO META-RAPS

As heurísticas construtivas têm como grande vantagem o fato de chegarem a soluções viáveis de boa qualidade em pouco tempo. No entanto, à medida que os problemas reais de transporte começam a contar com mais restrições, a qualidade destes procedimentos se deteriora. Para superar este problema podemos utilizar procedimentos meta-heurísticos que procuram soluções melhores iterativamente. Uma forma de se incrementar a qualidade de uma heurística construtiva transformando-a numa meta-heurística é através do método Meta-RaPS, cujos princípios forma detalhados no item 1.

Utilizando estes conceitos estabelecidos na revisão bibliográfica, pudemos reformular o algoritmo de Clarke & Wright visto na seção anterior de forma a garantir que seu desempenho fosse incrementado de uma maneira bastante simples. Na versão utilizada do Meta-RaPS, optamos por uma estrutura ainda mais simples de seleção da próxima atividade que requer apenas uma variável de controle, tornando mais fácil sua implementação e entendimento.

Definimos uma variável chamada de P_H que representa a probabilidade de num dado momento a regra heurística ser utilizada na junção de dois nós para formar uma rota. Esta variável é estabelecida no início do programa como um valor entre 0 e 1. A cada passo do algoritmo de Clarke & Wright o programa escolhe um número aleatório também entre 0 e1. Caso este número seja menor do que P_H a regra heurística é utilizada na formação da próxima rota. Caso este valor seja maior do que P_H, aquela atividade é ignorada e passamos à próxima atividade da lista.

Assim, quanto maior o valor de P_H, mais parecida com solução original proposta pela heurística de Clarke & Wright a solução formada tenderá a ser.

Analogamente, quanto menor este valor, maior a probabilidade das duas soluções encontradas serem diferentes. No caso extremo de $P_H = 1$, a solução obtida será idêntica a de Clarke & Wright. Além disso, optamos por não implementar um procedimento de melhoria para ser utilizado no final do processo, sendo este um potencial campo para pesquisas futuras.

Além desta modificação na estrutura básica da heurística construtiva, implementamos também algumas modificações no procedimento inicial de atendimento da demanda que ultrapassa a capacidade dos veículos, incorporando elementos aleatórios. O novo procedimento funciona da seguinte maneira:

- Para cada loja
    - Verifique qual o maior veículo que pode atender a loja
    - Enquanto a demanda remanescente for maior do que a capacidade deste veículo
        - Caso esta demanda seja maior do que a capacidade deste veículo somada a do segundo maior veículo capaz de atendê-la:
            - Envie um caminhão deste tipo com carga completa
        - Caso esta demanda seja maior do que a capacidade deste veículo somada a do terceiro maior veículo capaz de atendê-la:
            - Escolha aleatoriamente um dos dois maiores veículos
            - Envie um caminhão deste tipo com carga completa
        - Caso esta demanda seja maior do que a capacidade deste veículo somada a do quarto maior veículo capaz de atendê-la:
            - Escolha aleatoriamente um dos três maiores veículos
            - Envie um caminhão deste tipo com carga completa
        - Recalcule a demanda remanescente

assuma valores diferentes, mas sempre menores do que a capacidade do maior veículo que pode atendê-la. O fato da demanda que irá ser roteirizada assumir valores distintos altera o comportamento da heurística construtiva dando origem a soluções diferentes e potencialmente melhores.

5.3. ALGORITMO GENÉTICO

O passo seguinte na busca por melhores soluções para o problema abordado foi a implementação de um método meta-heurístico. Neste sentido, foi adotado o Algoritmo Genético. Na presente seção, iremos detalhar de que forma a teoria vista foi utilizada na construção deste método de solução, suas principais características e os resultados obtidos.

Um das etapas mais importantes na implementação de um AG é a escolha da estrutura de codificação de uma solução, ou o DNA de um indivíduo. Esta estrutura deve ser capaz de armazenar toda a informação necessária para representar de maneira precisa uma determinada solução.

No entanto, o material genético armazenado em um indivíduo não precisa necessariamente representar de forma direta a solução. A informação armazenada no DNA constitui o genótipo do indivíduo, enquanto a solução final representa o seu fenótipo. Para passar do genótipo para o fenótipo, o algoritmo conta com um conjunto de regras e procedimentos que lhe permite fazer esta transição, assim como na natureza, o material genético é convertido no ‘corpo’ do indivíduo.

Como estamos trabalhando com um problema de roteirização, a solução que desejamos obter é um conjunto de rotas que atenda a demanda das lojas de modo a minimizar o custo da operação respeitando suas restrições. Desta forma a estrutura da solução deve conter a informação necessária para a construção destas rotas. No entanto, neste problema, estamos abrindo a possibilidade de que uma loja seja visitada por vários caminhões mesmo que sua demanda possa ser atendida por apenas um. Assim, além de determinar a seqüência de lojas a ser atendida em cada rota, devemos determinar quantos

paletes serão entregues em cada uma delas. Para representar o problema estruturado desta forma, utilizamos a seguinte codificação:

Parâmetros:

N – número de indivíduos numa população;
R – número máximo de rotas em uma solução;
T – tamanho máximo de uma rota em número de lojas visitadas;
L – número de lojas a serem atendidas;
K – tipos de caminhões disponíveis.

Índices:

i – número do elemento na população, i = 1 - N;
j – número da rota na solução, j = 1 - R;
k – posição da loja na rota, k = 1 - T.

Variáveis:

Pop_{ijk} – loja visitada na posição k da rota j na solução i, Pop_{ijk} = 0 - L;
Pop_{ij0} – tipo de veículo utilizado para atender a rota j na solução i, Pop_{ij0} = 1 - K.

Esta estrutura de codificação nos permite representar em um vetor solução toda a informação necessária para o estabelecimento das rotas e dos veículos que irão percorrê-las, no entanto não representa a quantidade de paletes que será entregue para cada uma das lojas. Isto ocorre, pois optamos por manter a atribuição da demanda a cada loja como uma característica fenotípica determinada por um algoritmo simples de alocação. Desta forma mantemos reduzido o tamanho do vetor solução e garantimos um processo de evolução mais simples e rápido.

A alocação da quantidade entregue em cada parada de um veículo e determinada pelo seguinte algoritmo:

- Para cada rota de uma solução, começando da primeira:
    - Inicie com o caminhão vazio:
    - Para cada loja da rota, enquanto houver capacidade ociosa no veículo:
        - Se a demanda remanescente da loja for menor do que a capacidade ociosa:
            - Atenda toda a demanda da loja.
        - Caso contrário
            - Entregue toda a capacidade disponível e termine a rota

Obedecendo este algoritmo, podemos determinar, além da seqüência de lojas visitadas em cada rota, a quantidade entregue em cada parada. Este valor fica armazenado na variável Q_{ijk}.

Uma vez definida a estrutura básica de codificação, o próximo passo no AG é a geração de uma população inicial de indivíduos ou soluções. Neste trabalho estaremos adotando duas técnicas diferentes para elaboração destes indivíduos. Uma delas consiste na geração de valores aleatórios para cada posição de DNA do indivíduo, o que gera soluções iniciais muito ruins, mas garante uma ampla exploração do espaço de soluções possíveis. Outra abordagem é o uso de uma solução inicial obtida pela heurística de Clarke & Wright ou do Meta-RaPS. Este segundo método garante a convergência muito mais rápida do algoritmo, mas pode conduzir a ótimos locais e limitar sua capacidade de encontrar soluções melhores.

Outro elemento fundamental no algoritmo genético é o processo de seleção dos indivíduos que irão se recombinar para dar origem a nova população. Como foi exposto na teoria sobre AGs, existem muitos métodos para se fazer esta seleção, cada qual com vantagens e desvantagens. No desenvolvimento deste

trabalho, dois dos métodos mais populares foram implementados, sendo que um deles apresentou resultados significativamente melhores nos testes e por isso foi o escolhido.

O primeiro método testado consiste na atribuição a cada individuo de uma probabilidade de ser selecionado como “pai” proporcional a sua função utilidade. Assim indivíduos que apresentam melhores soluções para o problema possuem mais chances de se reproduzir, mas todos os indivíduos possuem alguma probabilidade. Este método é interessante, pois garante uma amplitude de recombinações bastante grande possibilitando o surgimento de indivíduos bem diversificados. No entanto, esta abordagem faz com que indivíduos ruins gerem muitos descendentes diminuindo o número de boas soluções em cada geração.

O segundo método considerado baseia-se na seleção apenas dos melhores indivíduos da população, para dar origem à geração seguinte. Para tanto, todos os indivíduos de uma população são classificados numa em ordem decrescente de acordo com sua utilidade e apenas os x melhores são escolhidos para formar o grupo de onde serão escolhidos os pais de cada elemento da próxima geração. x neste caso é um parâmetro de algoritmo que deve ser estabelecido pelo usuário. Cada elemento deste grupo tem igual probabilidade de ser escolhido.

Este segundo método, conhecido como elitismo, possibilita o surgimento de um maior numero de indivíduos de qualidade, e assim, maior velocidade na obtenção de boas soluções. Em todos os testes realizados este método apresentou melhores resultados e por isso foi adotado. A escolha do parâmetro x não apresentou impacto significativo no desempenho do algoritmo e seu valor foi fixado em 5 após alguns testes.

O passo seguinte nos AGs é a recombinação dos indivíduos selecionados (por qualquer dos métodos descritos). Para dar origem a uma nova população. Neste ponto os algoritmos genéticos apresentam uma forma significantemente diferente dos demais métodos de soluções de gerar novos indivíduos. Esta função e realizada por um operador chamado de crossover.

O crossover, assim como na biologia, permite o intercâmbio de material genético potencialmente útil na formação de novas soluções. Para controlar a taxa de utilização deste operador existe um parâmetro Pc que estabelece a probabilidade de que ele seja utilizado. Caso este operador não seja utilizado para combinar duas soluções para gerar uma terceira, é feita uma copia idêntica a um dos pais escolhidos aleatoriamente, o que representa uma reprodução assexuada. O crossover funciona da seguinte maneira:

- Caso seja escolhido crossover com probabilidade = Pc;
    - Escolha dois indivíduos para serem os pais (X e Y);
    - Escolha um ponto “z” no vetor solução para efetuar a quebra;
    - Para j=1 até z e k=0 até t;
        - Copie os valores de $popx_{j,k}$;
    - Para j=z+1 até r e k=0 até t;
        - Copie os valores de $popy_{j,k}$.

O processo de geração de novos elementos é seguido por outro operador baseado na biologia; a mutação. No contexto deste problema, foram estabelecidas três modalidades de mutação usadas em conjunto por modificar as soluções geradas pela recombinação. A primeira forma de mutação consiste na simples alteração da informação contida em um lócus ou variável do problema. Neste caso, um valor aleatório, representando uma loja ou um tipo de caminhão dependendo da posição do vetor é escolhido aleatoriamente com uma probabilidade Pm. Como visto na literatura e comprovado com os testes, esta probabilidade deve ser bastante baixa para garantir um bom desempenho.

Caso este valor seja muito alto, varias mutações ocorrem ao mesmo tempo destruindo soluções boas. No modelo o valor adotado para Pm foi 0,01%.

O segundo tipo de mutação implementado consiste na troca de posição de duas lojas numa mesma rota. Assim, o conjunto de lojas visitadas por um determinado caminhão é mantido, alterando apenas a seqüência na qual elas são atendidas. Esta operação permite que conjuntos promissores de lojas sejam colocados em uma seqüência que otimize seu custo de atendimento e tenham suas restrições de janela de tempo respeitadas.

O terceiro e último tipo de mutação proposto neste modelo é a troca de lojas de uma rota para a outra. Esta geração permite a formação de rotas atendendo conjuntos diferentes de lojas buscando sua otimização. Estes dois últimos tipos de mutação são implementados da seguinte forma de acordo com uma probabilidade Pt:

- Para cada variável $pop_{i,j,k}$ com uma probabilidade = Pt;
    - Escolha uma das duas formas de troca com 50% de probabilidade para cada;
        - Caso seja escolhida a troca na mesma rota aleatoriamente;
            - Escolha outra posição na rota aleatoriamente;
            - Inverta suas posições;
        - Caso seja escolhida a troca entre rotas;
            - Escolha outra rota aleatoriamente;
            - Escolha uma posição aleatória nesta rota;
            - Inverta suas posições.

Logo após o processo de geração da nova população, uma rotina simples de correção é aplicada às soluções obtidas para eliminar algumas características indesejáveis. Este procedimento percorre todas as rotas formadas em cada uma das soluções fazendo o seguinte:

- Retira da rota atendida por um determinado veiculo as lojas que não podem recebê-lo;
- Retira da rota as lojas que não estão recebendo nenhuma carga;
- Elimina a demanda entregue em posições vazias do vetor solução.

Tal rotina elimina problemas existentes nas soluções construídas a partir das recombinações e mutações garantindo um menor número de soluções inviáveis formadas a cada geração.

Uma vez que a nova população esteja totalmente constituída, o próximo passo no algoritmo é a avaliação de utilidade de cada um de seus indivíduos, que neste caso representa o custo total para atender a demanda.

Como já foi dito anteriormente, os custos de transporte são baseados em valores fixos preestabelecidos para cada localidade acrescidos de um delta para cada loja alem da primeira presente numa rota. A utilidade de um indivíduo será o somatório do custo de se percorrer todas as rotas que compõe a solução.

No entanto como o algoritmo genético baseia-se na recombinação e mutação aleatórias, são geradas constantemente soluções que não atendem as restrições do problema. Assim, para fazer com que o algoritmo convirja para soluções viáveis, atribuímos uma penalidade que é somada ao custo total de uma solução para cada restrição que não é atendida. Esta penalidade irá diminuir a utilidade daquelas soluções que não atendem todas as restrições, diminuindo assim sua probabilidade de gerar descendente. Por outro lado os indivíduos que respeitarem um maior número de restrições serão beneficiados e criarão um maior número de descendentes garantindo a convergência para soluções implementáveis.

Adotamos também o uso de penalidades variáveis de acordo com a quantidade de restrições desrespeitadas. Quanto maior o número de restrições desrespeitadas de uma determinada categoria, maior a penalidade atribuída a cada uma delas. Conseqüentemente, quando há um pequeno número de restrições não sendo atendidas, a penalidade marginal diminui. Este mecanismo permite que soluções fortemente inviáveis convirjam rapidamente para soluções viáveis, dado o alto custo associado às penalidades. Da mesma forma ele permite que soluções viáveis violem algumas restrições para explorar melhor a sua vizinhança na busca de melhorias.

Todos estes conceitos foram utilizados no calculo da função utilidade para as seguintes restrições:

- A demanda de uma loja não é totalmente satisfeita;
- A duração de uma rota ultrapassa a jornada máxima de trabalho;
- As janelas de recebimento não são respeitadas.

Para cada uma destas restrições foi estabelecida uma penalidade associada cujo valor relativo irá afetar o comportamento do algoritmo até que ele convirja para uma solução viável. Daí em diante, o valor destas penalidades não terá grande impacto em seu desempenho.

O critério de para utilizado no algoritmo genético foi um número fixo de iterações. Como a cada iteração várias soluções diferentes são testadas, este parâmetro pode afetar significativamente o desempenho do algoritmo. Caso um número pequeno de iterações seja definido, pode não haver tempo suficiente para que boas soluções sejam encontradas. Por outro lado, se forem permitidas muitas iterações, corremos o risco de desperdiçar muito tempo de processamento sem que melhorias sejam obtidas.

Para calibrar este parâmetro fizemos vários testes variando o número de iterações. O gráfico da Figura 5.2 apresenta a relação existente entre a qualidade da solução obtida e o número de iterações. Como o algoritmo genético trabalha com aleatoriedade, muitas vezes o resultado obtido para um mesmo número de iterações é diferente em duas rodadas. Mesmo assim, a correlação entre as duas variáveis é bastante alta.

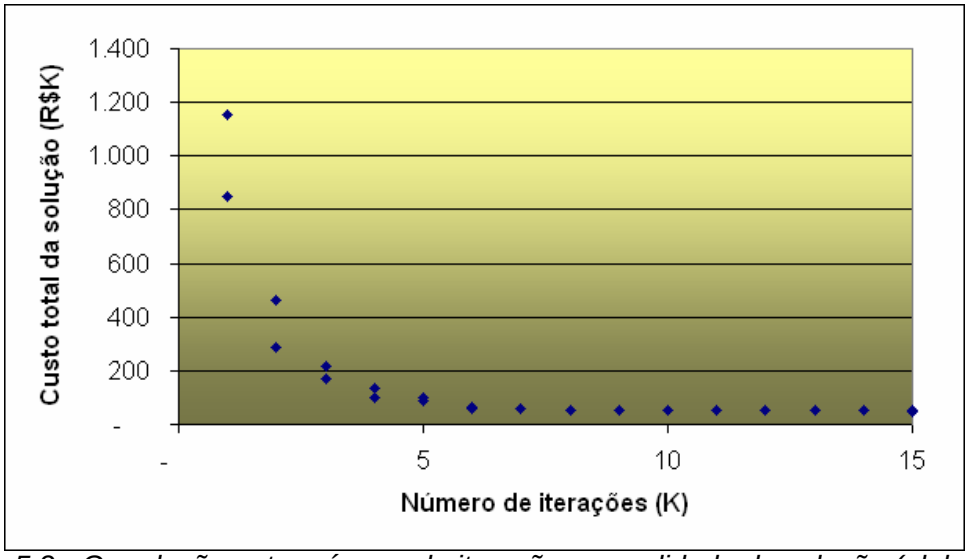

Figura 5.2 - Correlação entre número de iterações e qualidade da solução (elaborado pelo autor)

Outro ponto que chama a atenção no gráfico e que a partir de certo número de iterações, a qualidade da solução não melhora muito. Na prática, optamos por estabelecer um limite de 10.000 iterações como critério de parada para o algoritmo.

5.4. ALGORITMO GENÉTICO PARALELO

Passaremos agora a descrever como o algoritmo genético foi paralelizado, a forma como foi implementado e os resultados obtidos. Discutiremos os aspectos mais relevantes e todas as adaptações feitas no algoritmo para que este funcionasse em paralelo.

Conforme foi visto na Seção 2.4, existem duas maneiras básicas de se implementar um algoritmo genético de forma paralela, uma delas trabalhando com uma única população que tem a avaliação dos indivíduos distribuída entres os processadores disponíveis (método global), e outra onde cada nó recebe uma sub-população que eventualmente troca indivíduos com as outras (método das ilhas). Neste trabalho adotamos o segundo método por ele permitir ao algoritmo explorar um universo maior de indivíduos possibilitando o aparecimento de soluções melhores num tempo razoável e diminuindo a probabilidade de que o mesmo convirja para ótimos locais.

A adaptação da versão seqüencial do algoritmo para sua forma paralela foi feita com o auxílio de funções de comunicação entre computadores disponíveis na biblioteca padrão MPI cujos detalhes são discutidos no Anexo A. Assim como os outros métodos de solução utilizados neste trabalho, todo o algoritmo foi programado na linguagem C. O código fonte completo para todos os algoritmos encontram-se no Anexo D.

A paralelização ocorre da seguinte maneira: no início da execução, um dos nós do cluster com o qual o usuário tem interface (que chamaremos de nó 0) faz a leitura do arquivo que contém os dados do problema e em seguida aciona os demais nós através do comando MPI_Init() (vide Anexo A). A partir daí cada nó executa um comando para saber quantos nós compõem a rede e qual é o seu número neste conjunto.

Deste momento em diante, cada nó atua de maneira independente realizando todas as etapas do algoritmo genético tradicional. Quando o algoritmo atinge um número pré-estabelecido de gerações é executada uma rotina que realiza a migração dos melhores indivíduos da população de um determinado nó para um nó adjacente. Assim, caso existam 5 nós (0, 1, 2, 3, 4) os indivíduos migrarão de 0 para 1, de 1 para 2, ... e de 4 para 0 completando o ciclo. Este

procedimento é repetido toda a vez que o número de gerações alcançar este número fixado de gerações.

No final, todos os nós enviam sua melhor solução para o nó 0 que irá compará-las e exibir como resultado final a solução que apresentar o menor custo total.

A característica do AG Paralelo de trabalhar com múltiplas populações simultaneamente permite uma ampla exploração do espaço de soluções aumentado as chances de se obter boas soluções. Ao mesmo tempo, a migração de indivíduos de uma população para outra garante que nenhum nó perca muito tempo trabalhando com uma população cujo resultado seja ruim, pois esta será melhorada com a chegada de indivíduos de populações vizinhas. Além disso, o fato de cada nó trabalhar a maior parte do tempo de forma independente acarreta num baixo tráfego de rede proporcionando *speedups* bastante elevados.

Combinando estas características com a capacidade de processamento fornecida pelos 60 nós do Cluster no Tanque de Provas Numérico da Naval (Anexo A) obtivemos resultados muito animadores, conforme será explorado no próximo capítulo.

6. ANÁLISE DOS RESULTADOS

6.1. SOLUÇÃO ATUAL DA EMPRESA

O primeiro passo na análise dos resultados obtidos pelos métodos propostos é a definição de um termo de comparação. Para tanto foi feita uma ampla análise dos resultados atuais obtidos pela empresa no processo de abastecimento. Nesta fase, alguns pontos chamaram a atenção e merecem ser discutidos com mais detalhes para que a base de comparação seja compatível com o resultado do modelo proposto.

Ao analisar os valores pagos pela empresa para as transportadoras pudemos verificar que os mesmos diferem daqueles que seriam de se esperar com base nos contratos, conforme pode ser observado na Tabela 4.3. Em alguns casos, existem valores pagos na prática muito mais baixos que o esperado. Estas diferenças decorrem principalmente de negociações pontuais com alguns fornecedores e de ajustes, como quando, por exemplo, uma parcela do frete de um dia é pago em outro. Por este motivo não poderemos utilizar os custos históricos para comparar com os resultados obtidos pelo modelo, pois estes não são calculados de forma igual, tornando necessária uma outra abordagem para comparação.

Outro ponto que logo chama a atenção quando estudamos o histórico das entregas efetuadas, é que muitas vezes a capacidade dos caminhões parece ser violada. Isto ocorre na prática, pois as vezes, é possível remontar alguns paletes e empilhá-los de modo que o volume ocupado seja menor. Para tornar os resultados comparáveis, tendo em vista que este procedimento não pode ser adotado no modelo, é necessário corrigir este efeito nos dados históricos.

Também é importante ressaltar que os valores de demanda utilizados como dados de entrada para o modelo foram obtidos a partir do histórico de entregas. Assim, os roteiros utilizados pela empresa estão perfeitamente adequados à demanda que estamos adotando no trabalho. No entanto, está demanda não necessariamente reflete os pedidos que de fato foram feitos à Central de Programação. Isto ocorre, pois, na prática, uma parte da demanda de um dia pode ser deixada para o dia seguinte caso se obtenha maior eficiência no transporte.

Levando-se em conta todos estes fatores, foi necessário adotar alguns procedimentos para criar uma base de comparação para o resultado dos modelos a partir do histórico das operações do grupo. Isto foi feito da seguinte forma:

- Em primeiro lugar, o valor histórico pago para as transportadoras foi ignorado, pois este não seria comparável;
- O custo foi então calculado a partir das entregas efetivamente realizadas com base nos custos contratuais utilizando-se exatamente as mesmas regras adotadas no modelo;
- No caso do volume entregue na prática em um roteiro ser maior do que a capacidade do caminhão utilizado, adotamos que este caminhão teve que retornar ao CD antes de prosseguir com as entregas;
- Se o volume for maior do que a capacidade mesmo que apenas uma loja seja atendida, adotamos que um veículo maior teve que ser utilizado.

Com todos estes ajustes pudemos finalmente obter uma solução original tecnicamente comparável com o resultado dos modelos propostos neste

trabalho. Os custos após os ajustes dos 7 cenários básicos que estamos analisando podem ser observados na tabela a seguir:

Caso	Custo original
Dom.	R$ 21.677
2ª	R$ 35.766
3ª	R$ 79.572
4ª	R$ 44.138
5ª	R$ 61.801
6ª	R$ 71.388
Sab.	R$ 66.470

Tabela 6.1 - Custos ajustados dos cenários (elaborado pelo autor)

6.2. COMPARAÇÃO ENTRE OS MÉTODOS

Os métodos de solução descritos no capítulo anterior foram aplicados à resolução do problema de abastecimento das lojas do Grupo para um conjunto de dados selecionado representando um dia típico de operação. O dia foi escolhido juntamente a gerência da empresa por ser uma data representativa para as operações no restante do ano. Além desta data básica (uma 4ª feira) foram analisados outros 6 dias na mesma semana para verificar o impacto na qualidade das soluções decorrentes da mudança no volume, tendo em vista que nos finais de semana este é, normalmente, bem menor. A tabela a seguir apresenta um resumo dos 7 casos considerados.

Caso	Número de lojas	Demanda total (Qtd. Paletes)
Dom.	48	1.228,0
2ª	158	1.997,0
3ª	254	4.094,5
4ª	214	2.269,0
5ª	242	3.104,5
6ª	229	3.999,0
Sab.	206	3.185,5

Tabela 6.2 - Resumo dos casos considerados (elaborado pelo autor)

Como o número de lojas que estamos considerando nestes problemas é muito grande, as soluções obtidas apresentam inúmeros roteiros, o que torna inviável a representação gráfica dos mesmos. Para não exibir uma quantidade enorme de tabelas com os resultados, optamos por mostrá-los de forma sintética por meio de alguns indicadores chave da qualidade das soluções. O resultado completo para o cenário base, resolvido pelo método do algoritmo genético paralelo encontra-se no Anexo C. Apenas para ilustrar o tipo de solução gerada pelo método, e permitir algumas discussões, a Figura 6.1 apresenta alguns do roteiros formados.

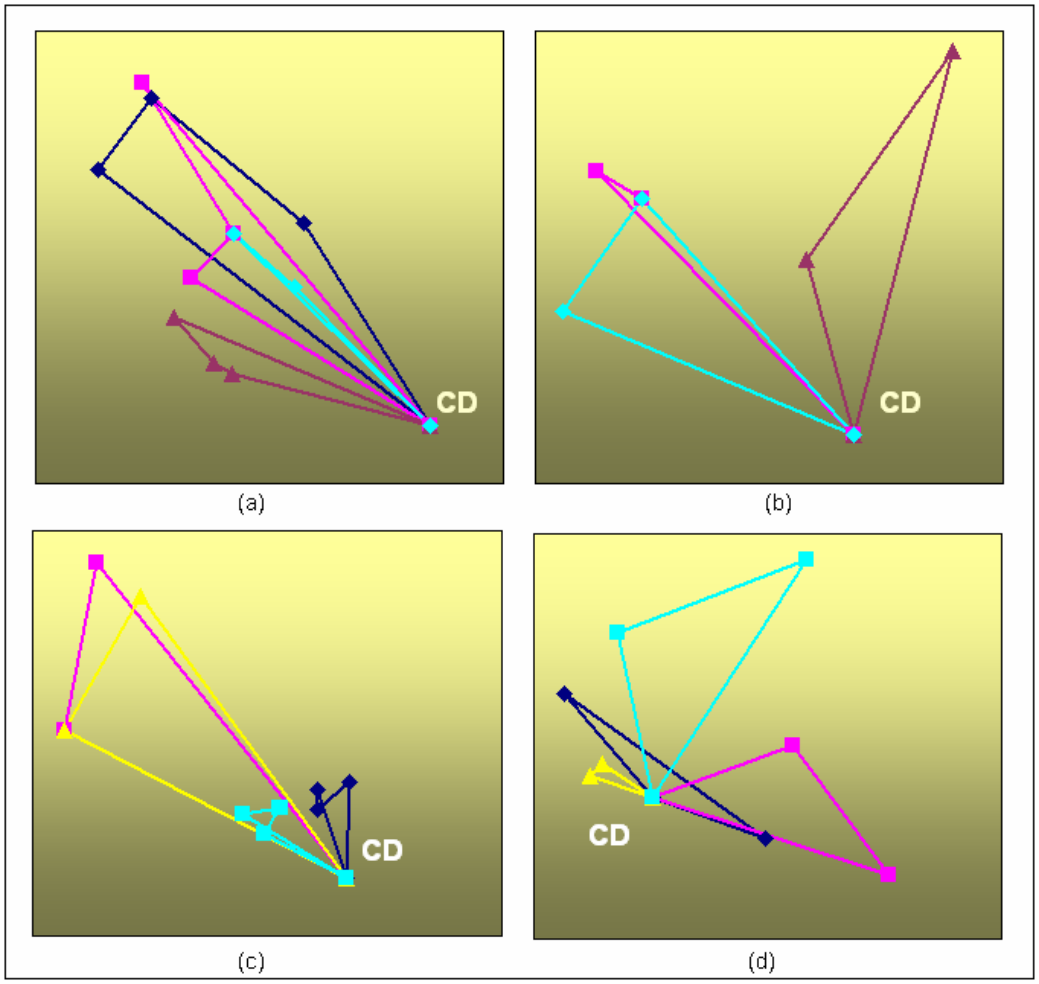

Figura 6.1 - Exemplo de roteiros formados pelo AG Paralelo (a, b) e pelo Meta RaPS (c, d) para o cenário base (elaborado pelo autor)

O ponto que mais chama a atenção quando comparamos os roteiros formados pelos dois métodos é que no Meta-RaPS, cada rota tende a ser composta por lojas situadas numa mesma distância do CD. Isto ocorre, pois o método de formação das rotas obtém as maiores economias no custo quando o frete para as duas lojas é o mesmo, ou pelo menos parecido.

Os resultados obtidos pelos métodos descritos no capítulo anterior foram razoavelmente diferentes entre si. A Tabela 6.3 apresenta um resumo do custo total do processo de abastecimento para cada um dos dias estudados.

Método	Custo total (R$)						
	Dom.	2ª	3ª	4ª	5ª	6ª	Sab.
Original	21.677	35.766	79.572	44.138	61.801	71.388	66.470
Clarke & Wright	22.143	30.071	71.431	37.129	55.680	65.822	60.518
Meta RaPS	21.976	29.747	71.255	36.759	55.335	65.494	59.794
AG	22.815	32.760	78.331	40.255	60.465	70.196	65.327
AG Paralelo	21.407	30.564	74.334	37.534	56.831	67.586	62.585

Tabela 6.3 - Custo total das soluções (elaborado pelo autor)

Como podemos perceber, no primeiro caso (Domingo), o AG Paralelo foi o que apresentou o melhor resultado. No entanto, para os demais casos, o método que proporcionou os custos mais baixos foi o Meta-RaPS. É importante notar também que em todos os casos, o custo das soluções propostas é inferior ao praticado atualmente pela empresa.

Como pode ser observado na Tabela 6.4, os métodos de solução propostos, além de proporcionar uma melhoria significativa no custo total da operação, garante que um número bem menor de restrições sejam violadas. Desta forma, minimizamos os problemas de excesso de carga transportada nos caminhões, jornadas muito longas para os motoristas e desrespeito às janelas de recebimento, viabilizando não só uma melhoria nos custos como também no nível de serviço proporcionado e na segurança das operações.

Método	Restrições violadas			
	Jornada	Falta	Janela	Capacidade
Original	8	0	?	43
Clarke & Wright	8	0	0	0
Meta RaPS	7	0	0	0
AG	7	1	1	0
AG Paralelo	6	1	0	0

Tabela 6.4 - Restrições violadas nas soluções para o caso base (elaborado pelo autor)

A restrição de jornada de trabalho é violada algumas vezes em todos os casos, pois algumas lojas simplesmente ficam tão longe que não podem ser atendidas no tempo disponível. A falta é zero na solução original, pois os próprios paletes entregues foram considerados a demanda para o modelo. No entanto, sabemos que na prática muitas vezes o grupo deixa de entregar alguns produtos, enviando-os apenas no dia seguinte. Nos dois AGs, um palete deixou de ser entregue pois esta é uma restrição relaxada à qual atribuímos uma penalidade. No Clarke & Wright e no Meta-RaPS isto não é possível pois a restrição é rígida.

As janelas de recebimento, que também foram relaxadas nos dois tipos de AGs, foram respeitadas em quase todos as rotas e métodos. Apenas no AG, uma loja teve um atraso de 6 min. na entrega. Não foi possível comparar com a solução original, pois não sabemos o horário em que as entregas foram feitas na prática. Finalmente, a capacidade do caminhão é uma restrição rígida em todos os modelos, e, portanto, foi sempre respeitada. Todavia, na solução original, 43 caminhões são enviados com mais paletes do que sua capacidade teórica. Isto se deve a remontagem de paletes no momento do carregamento para ocupar espaços vazios no veículo. Apesar de ser uma prática comum, este fato não pode ser incorporado no modelo.

O tempo médio de processamento em cada rodada varia bastante de acordo com método de solução utilizado. Para os algoritmos genéticos, este tempo é diretamente proporcional ao número de iterações, e varia de acordo com o

número de lojas a serem atendidas. No caso da heurística construtiva ele depende apenas do tamanho do problema, já que é executado em apenas uma iteração. Finalmente para o Meta-RaPS, este tempo depende do número de iterações e do tamanho do problema, mas é significativamente menor do que no caso dos genéticos. A tabela a seguir apresenta o tempo gasto na resolução dos modelos para cada um dos métodos e dias da semana.

Método	Tempo de processamento (s)						
	Dom.	2ª	3ª	4ª	5ª	6ª	Sab.
Clarke & Wright	0,6	3,1	20,0	9,0	14,4	12,2	9,0
Meta RaPS	12,2	111,7	392,8	265,8	361,3	291,4	231,7
AG	388,8	1.398,0	1.888,8	1.342,8	1.581,0	1.808,4	1.533,6
AG Paralelo	627,0	1.019,0	2.698,0	1.503,0	1.439,0	1.312,0	1.113,3

Tabela 6.5 - Tempos de processamento (elaborado pelo autor)

A análise da Tabela 6.5 deixa claro que o Clarke & Wright é o método mais rápido em todos os casos. No entanto, o Meta-RaPS apresenta um tempo médio de processamento bastante razoável, deixando um intervalo de tempo suficiente para que o analista possa estudar as soluções e fazer os ajustes necessários.

A solução proposta pelo modelo, independentemente do método utilizado, consiste num conjunto de rotas, cada uma sendo percorrida por um determinado tipo de veículo. Estas rotas podem ser entregas diretas, caso apenas uma loja seja visitada, ou roteiros, que possuem mais de uma loja.

Os diferentes métodos de solução utilizados apresentaram resultados diferentes não só em termos do custo total da operação, mas também quanto ao mix de veículos utilizados e a proporção de entregas diretas. A Figura 6.2 ilustra estas diferenças para o nosso cenário base.

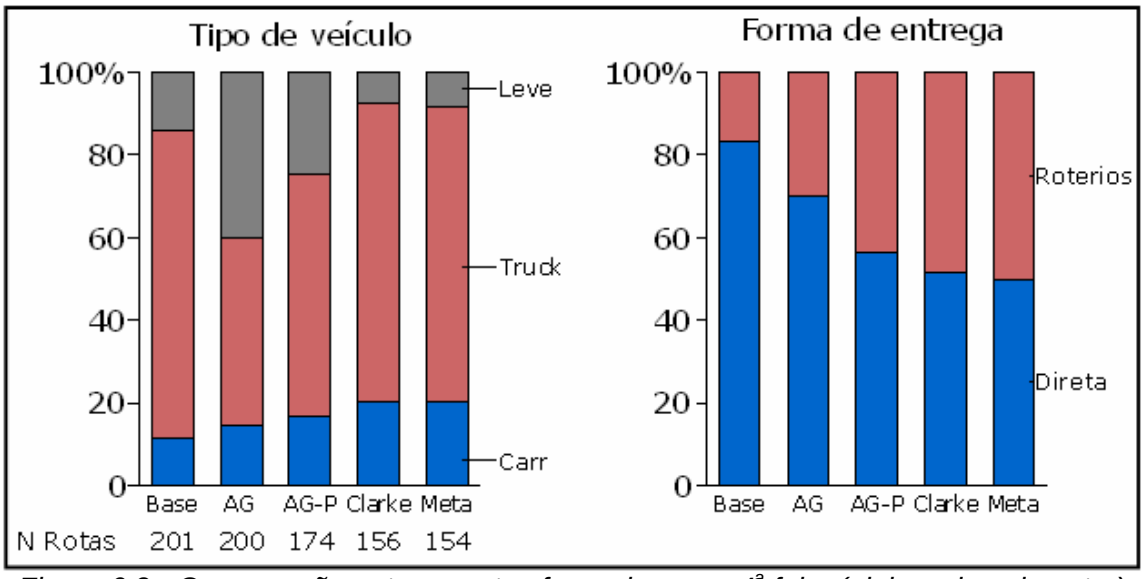

Figura 6.2 - Comparação entre as rotas formadas para 4ª feira (elaborado pelo autor)

Quando comparamos as soluções fornecidas pelo algoritmo genético seqüencial e o paralelo, podemos notar que, a medida que o segundo obteve resultados melhores, um número maior de roteiros foi utilizado em detrimento das entregas diretas. Além disso, a proporção de veículos grandes (carretas e trucks) é maior no paralelo, fato que pode ser explicado pelo menor custo unitário proporcionado por estes veículos.

Por outro lado, quando comparamos os resultados obtidos pelo Clarke & Wright e pelo Meta-RaPS não percebemos grandes diferenças entre eles. Isto ocorre por que a estrutura de formação de soluções de ambos é muito parecida e, apesar do Meta-RaPS obter soluções de melhor qualidade, estas não diferem substancialmente daquelas obtidas pelo CW.

Finalmente, quando comparamos o Clarke & Wright ou o Meta-RaPS com os genéticos, podemos perceber que tanto o número de entregas diretas é menor, quanto a utilização de veículos de maior capacidade é mais significativa, o que em parte explica os melhores resultados obtidos por estes métodos na maioria dos casos

O fato das soluções propostas em geral proporcionarem resultados que utilizam um menor número de roteiros é que irá permitir a geração de economias de longo prazo. Isto ocorre, pois com os novos roteiros é possível redimensionar a frota dedicada através de mudanças nos contratos garantindo tanto economias para a empresa como a sobrevivência das transportadoras.

A economia média ponderada pelo volume da operação obtida pelo método Meta-RaPS, foi de aproximadamente 11% ao longo da semana. Se projetarmos este resultado para os demais dias do ano, considerando o custo total anual de aproximadamente R$13,2 milhões, chegamos a uma economia de R$1,4 milhões apenas na operação do CD abordado neste trabalho. Dadas as proporções da operação de abastecimento das unidades do Grupo, cada ponto percentual de melhoria operacional representa R$132 mil de economia anual. Este fato por si só já justifica o investimento em métodos mais eficazes de roteirização.

Ao contrário do que imaginávamos no princípio do trabalho, o fato dos algoritmos genéticos conseguirem ignorar a restrição de que cada loja só pode ser atendida por um veículo não proporcionou, na maioria dos casos, resultados melhores que o Meta-RaPS. Isto se deve principalmente a dois fatores. Em primeiro lugar, com a eliminação desta restrição, o espaço de soluções viáveis para o problema cresce demais, devido ao número de novas combinações permitidas. Desta forma, o algoritmo não consegue convergir para uma solução muito boa. O segundo fator que prejudica o desempenho do algoritmo é a necessidade de se calcular a quantidade entregue em cada parada já que isto não é imediato como quando adotamos a restrição de uma visita por loja (neste caso, toda a demanda da loja deve ser atendida em cada parada). Este cálculo extra exige um grande esforço computacional e torna o algoritmo bem mais lento.

Tal fato mostra que a hipótese simplificadora que impede múltiplas entregas numa mesma loja, apesar de restringir as soluções possíveis, é muito boa no sentido de garantir que os algoritmos convirjam mais rapidamente para soluções de qualidade.

Se por um lado o AG Paralelo não apresentou melhorias significativas quando comparado ao Meta-RaPS, por outro pudemos perceber que os ganhos versus a versão seqüencial são expressivos (em torno de 5,5%). Assim, se as técnicas de computação paralelas puderem ser aplicadas em algoritmos que já apresentem em suas versões seqüenciais resultados de alta qualidade, sem dúvida os ganhos obtidos, tanto em tempo de processamento, como na qualidade final das respostas, certamente serão consideráveis.

Um ponto importante, que não pode deixar de ser comentado neste momento, é falta de conexão entre os valores contratuais pagos pelo grupo para as empresas transportadoras e os drivers de custo reais da operação de transporte. Como já discutimos anteriormente, o frete pago é definido de acordo com a região onde a loja se situa além de um adicional para cada loja extra visitada. Este valor não leva em conta a distância total percorrida nem o tempo do trajeto, que são os fatores geradores de custo para a transportadora. Deste modo, uma solução que minimiza o valor pago pela empresa não necessariamente reduz os custos da operação para a transportadora.

Para se precaver deste custo operacional mais elevado, as transportadoras têm que cobrar um valor mais alto do que seria possível caso a otimização fosse conduzida considerando-se os custos reais de transporte. Assim, se fosse possível alterar os contratos para que estes fossem baseados nos drivers reais de custo, poderíamos negociar condições mais favoráveis para nossos fornecedores e consequentemente baixar o custo total do processo gerando maior valor ao longo da cadeia produtiva.

6.3. IMPLEMENTAÇÃO DA SOLUÇÃO

Até agora discutimos o problema de roteirização dos caminhões no processo de abastecimento com uma abordagem bastante conceitual, focando o desenvolvimento de modelos e sua solução através de diferentes métodos. Estas análises nos permitiram concluir que as idéias propostas, se implementadas, têm grande potencial para gerar economias no processo e ganhos de produtividade. No entanto, o trabalho não termina com a resolução dos modelos em um computador. É preciso fazer com que eles se tornem parte da operação da empresa e sejam capazes de gerar boas soluções em todos os casos, com flexibilidade para lidar com situações específicas. Nesta seção estaremos explicando como os modelos se integram ao processo de abastecimento e as mudanças necessárias neste processo para que os mesmos possam ser utilizados na prática.

O novo processo, para absorver os modelos propostos anteriormente, deve ser ligeiramente diferente daquele apresentado no Item 1. A figura a seguir apresenta de forma simplificada o processo proposto:

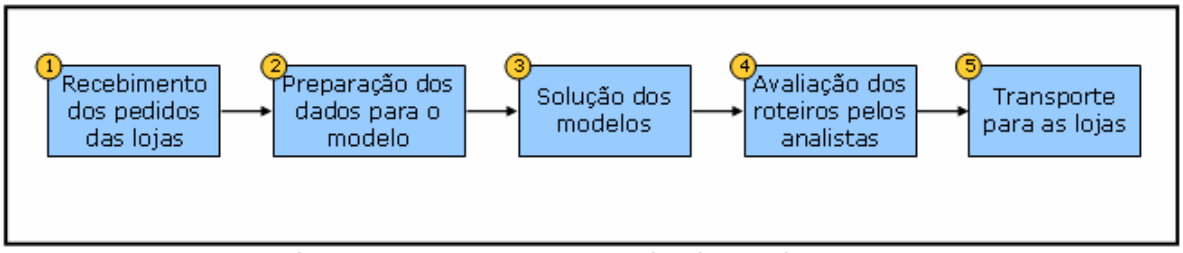

Figura 6.3 - Novo processo de abastecimento

Até o momento em que os pedidos chegam a central de programação não estamos propondo nenhuma alteração no processo. Porém, a partir do recebimento dos pedidos, são necessárias algumas mudanças nos procedimentos.

O primeiro passo para a resolução dos modelos é a criação dos arquivos que contém os dados de entrada. Este procedimento é bastante simples tendo em vista que um único arquivo no formato txt é utilizado. A única ação necessária por parte dos operadores é fazer a consolidação da demanda de todas as lojas a cada dia e atualizar o cadastro caso novas lojas sejam inauguradas (ou fechadas). Seria possível inclusive automatizar este procedimento via sistema, sem grandes dificuldades, o que tornaria o processo ainda mais eficiente.

Uma vez tendo sido gerados os arquivos de entrada, o modelo se encarrega de criar os roteiros apropriados para atender a demanda específica daquele dia, independentemente do método de solução que venha a ser utilizado. Como resultado, são gerados arquivos de saída especificando a seqüência de lojas que cada um dos roteiros deve atender, bem como os caminhões que irão percorrê-los e o horário de cada parada.

Para que o processo seja 100% capaz de se adequar a todas as restrições práticas é fundamental que um analista reveja as soluções propostas pelo modelo. Isto se dá, pois podem ocorrer situações imprevistas, como pedidos emergenciais, acidentes com caminhões, vias interditadas, etc. que não podem ser consideradas no modelo. Desta forma o analista, com base na sua experiência, poderá adaptar os roteiros criando soluções viáveis em todos os casos.

Uma vez que os roteiros estejam prontos, o processo de carregamento dos caminhões e transporte físico ocorre exatamente como era antes, concluindo o abastecimento.

Apesar de na teoria a metodologia proposta poder funcionar sem grandes problemas, proporcionando resultados de boa qualidade, na prática, ela ainda não foi testada. Neste sentido, o próximo passo lógico seria realizar um teste

piloto com a demanda de um dia para validar o modelo, bem como sua viabilidade de implementação.

A maior mudança, no entanto, não está no fluxo do processo, mas sim no conceito utilizado para a formação dos roteiros. Hoje, existem alguns roteiros pré-definidos que são utilizados para formar as rotas de entrega. Estes roteiros ficam armazenados no sistema e são utilizados sistematicamente para se planejar as entregas, independentemente da demanda específica do dia. Desta forma, apesar de proporcionar boas soluções na maioria dos casos, o método atual não permite que as rotas sejam otimizadas diariamente. O método que estamos propondo garante que as peculiaridades dos pedidos de cada dia sejam levadas em conta na formação dos roteiros. Assim, podemos trabalhar com rotas muito diferentes das usadas atualmente, o que pode gerar desconforto tanto para os analistas como para os motoristas que já estão acostumados com os roteiros usados hoje em dia.

Para garantir que o novo método seja utilizado na prática é fundamental assegurar que todos os funcionários envolvidos no processo de abastecimento estejam a par das economias e ganhos de produtividade proporcionados, fazendo com que eles passem a aceitar as mudanças e não se oponham à sua utilização.

Do ponto de vista prático, pudemos observar que o Meta-RaPS é um método que apresenta uma ótima relação custo-benefício, dado que ele é facilmente implementável podendo ser rodado em um único computador, e apresenta bons resultados. Já o algoritmo genético paralelo consegue obter as melhores soluções apenas nos problemas menores, e sua implementação exige a utilização de um cluster que representa um investimento considerável, além de acarretar em custos de manutenção e depreciação significativos.

Assim, podemos concluir que num primeiro momento o mais indicado seria a adoção do método Meta-RaPS. Caso este tenha uma boa aceitação e proporcione os resultados esperados, haverá um tempo para que o modelo seja perfeitamente ajustado às condições práticas. Assim, poderemos avaliar se os ganhos adicionais proporcionados por métodos mais sofisticados como o AG Paralelo justificam o investimento em um cluster.

É válido lembrar que, caso o cluster seja adotado, sua utilização estará restrita à uma hora por dia, o que garante muito tempo livre para que outras aplicações sejam desenvolvidas. Conforme visto no item 1, os clusters podem ser usados para uma série de aplicações empresariais como simulações financeiras, modelagem de processos, etc. o que pode gerar outros benefícios que justifiquem o investimento.

7. CONCLUSÕES

O presente trabalho apresentou um método científico para a resolução do problema de abastecimento das lojas do grupo varejista resultando em melhorias significativas em relação ao processo atual. Estas melhorias puderam ser obtidas graças à formulação do problema de modo a incorporar as restrições operacionais enfrentadas na prática, e à combinação de diferentes técnicas de solução buscando aproveitar as melhores características de cada uma delas.

Além de apresentar uma forma estruturada de resolução de problemas de roteamento de veículos, o trabalho introduziu o conceito de Computação Paralela que vem sendo cada vez mais utilizado em diversas áreas do conhecimento, e como o mesmo pode ser implementado na solução deste tipo de problema. De fato, a combinação de algoritmos eficientes, com a capacidade de processamento proporcionada pela computação paralela garante a obtenção de soluções muito boas num intervalo de tempo compatível com as necessidades da operação.

Este trabalho permite concluir que a utilização de heurísticas construtivas é uma forma simples e rápida de se obter melhorias nas soluções quando comparadas àquelas obtidas empiricamente. A adoção de meta-heurísticas mais sofisticadas, no entanto, é uma abordagem interessante, pois permite encontrar soluções ainda melhores. Este fato é particularmente útil quando a empresa possui operações de grande porte, onde cada pequeno ganho percentual na qualidade da solução pode significar grandes quantias no resultado financeiro da mesma.

A heurística construtiva de Clarke & Wright se mostrou uma ferramenta extremamente prática, seja pela simplicidade conceitual do método das

economias e facilidade de implementação, flexibilidade para se adaptar a diversas restrições operacionais, e pelo baixo tempo de processamento. Outro ponto importante é a possibilidade de obter rápidas melhorias nesta heurística através do algoritmo Meta-RaPS que a transforma numa meta-heurística pela introdução de elementos aleatórios no processo de formação de rotas. Os algoritmos genéticos que permitem a realização de buscas mais amplas no espaço de soluções viáveis e o uso de clusters para a paralelização do código e aumento do poder de processamento se mostraram ferramentas bastante interessantes, mas que na prática não conduziram a resultados significativamente melhores.

Finalmente, é importante indicar que os métodos utilizados neste trabalho podem ser o objeto de novas pesquisas para que sejam ainda mais aprofundados e adaptados a novos problemas práticos. No caso específico da empresa em questão, estes mesmos métodos podem ser adaptados para sua futura utilização em diferentes categorias de produtos, tais como frutas, legumes e verduras, carga refrigerada, etc. Além disso, todo trabalho aqui apresentado pode ser implementado com maior abrangência geográfica incluindo todos os centros de distribuição e lojas do grupo no Brasil. Alguns novos critérios e restrições também poderiam ser adicionados em futuras implementações, dentre eles principalmente a priorização de pedidos.

Além disso, como foi visto na seção 6.1, nem todos os aspectos práticos da operação de abastecimento puderam ser incorporados ao modelo. O principal deles, que poderia vir a ser considerado em trabalhos futuros é a remontagem de paletes. Este procedimento permite que um caminhão que ainda não tenha atingido o limite de peso, possa receber paletes extras, que são remontados para ocupar o volume ocioso. Apesar de ser mais demorado, este procedimento permite uma ocupação melhor do veículo e a conseqüente redução dos custos, podendo ser considerado em estudos futuros.

BIBLIOGRAFIA

BALLOU, RONALD H. **Gerenciamento da Cadeia de Suprimentos**: Planejamento, Organização e Logística Empresarial. São Paulo. Bookman, 2001.

CANTÚ-PAZ, ERICK **Implementing Fast and Flexible Parallel Genetic Algorithms**, Practical Handbook of Genetic Algorithms v.3, p.65-81, 1999.

CLARKE, G.; WRIGHT, J.W. **Scheduling of Vehicles from a Central Depot to a Number of Delivery Points**. Operations Research, v.12, p.568-581, 1964.

CORDEAU, J-F.; GENDREAU, M.; LAPORTE, G. **A Guide to Vehicle Routing Heuristcs**. Journal of Operational Research Society, n.53, p.512-522, 2002.

CUNHA, CLÁUDIO B. **Uma Contribuição para o Problema de Roteirização de Veículos com Restrições Operacionais**. Tese de Doutorado – Escola Politécnica da Universidade de São Paulo, 1997.

FISHER, M.; JAIKUMAR, R. **A Generalized Assignment Heuristics For Vehicle Routing**. Networks, v.11, p.109-124, 1981.

FUH-HWA, L.; SHENG-YUAN, S. **A Method for Vehicle Routing Problem with Multiple Vehicle Types and Time Windows**. Department of Industrial Engineering and Management – National Chiao Tung University, 1999.

GOLDEN, B.L.; ASSAD, A.A. **Vehicle Routing:** Methods and Studies. Elsevier Science Publishers B.V., Amsterdam, 1988.

HAJRI-GABOUJ, S.; DARMOU, S. **A Hybrid Evolutionary Approach for a Vehicle Routing Problem with Double Time Windows for the Depot and Multiple Use of Vehicles**. Institut National des Sciences Appliquées et de Technologie, 2003.

HWANG, K.; Briggs, F. A. **Computer Architecture and Parallel Processing**. McGraw-Hill International Editions, 1984.

LUNA, H. P. L.; GOLDBARG, M. C. **Otimização Combinatória e Programação Linear**. 5 ed. Rio de Janeiro: Editora Campus, 2000.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge: The MIT Press, 1996.

MIURA, M. **Resolução de um Problema de Roteamento de Veículos em uma Empresa Transportadora**. Trabalho de Formatura (Graduação) – Escola Politécnica, Universidade de São Paulo. São Paulo, 2003.

MORAGA, R. et al. **Solving the Capacitated Vehicle Routing Problem Using the Meta-RaPS Approach**. Industrial Engineering and Management Systems Department.

NAVAUX, P. O. A. **Introdução ao Processamento Paralelo**. RBC - Revista Brasileira de Computação, v.5, n.2, p.31-43, Outubro, 1989.

OCHI, L.; VIANNA, D.; DRUMMOND, L. **A Parallel Evolutionary Algorithm for the Vehicle Routing Problem with Heterogeneous Fleet**. PGCC – Universidade Federal Fluminense.

OMBUKI, B.; ROSS, B. J.; HANSHAR, F. **Multi-objective Genetic Algorithms for Vehicle Routing Problem with Time Windows**. Department of Computer Science - Brock University, 2004.

PARREIRAS, L. **Modelo Genético-Neural de Gestão de Carteira de Ações**. Trabalho de Formatura (Graduação) – Escola Politécnica, Universidade de São Paulo. São Paulo, 2003.

PELIZZARO, CLÁUDIA **Avaliação do Desempenho do Algoritmo de um Programa Comercial para Roteirização de Veículos**. Tese de Mestrado – Escolha de Engenharia de São Carlos, 2000.

PEREIRA, F.; TAVAREZ, J. **GVR: a New Genetic Representation for the Vehicle Routing Problem**, Instituto Nacional de Engenharia de Coimbra, 2002.

PITANGA, M. **Computação em Cluster**. 2004. Disponível em: <www.clubedohardware.com.br/cluster.html>. Acessado em: 16 ago. 2004.

SANTANA, R.; SANTANA, M. **Computação Paralela**. Departamento de Ciências de Computação e Estatística - USP São Carlos, 1997.

SOLOMON, M. M. **Algorithms for the Vehicle Routing and Scheduling Problems with Time Windows Constraints**. Operations Research, v.35, n.2, p.254-265, 1987.

THANGIAH, SAM R. **A Hybrid GA, Simulated Annealing and Tabu Search Heuristics for VRP with TW**. Practical Handbook of Genetic Algorithms v.3, p.347-376, 1999.

WINSTON, W. L. **Operations Research:** Applications and Algorithms. California: Duxbury Press, 1994.

ANEXOS

ANEXO A – Biblioteca MPI

As bibliotecas de programação mais utilizadas para trabalhar com clusters do tipo Beowulf, dentre as diversas opções disponíveis, são:

OpenMP

O OpenMP tem como objetivo prover a comunicação entre processadores com memória compartilhada, ou em máquinas que simulem memória compartilhada em cima de memória distribuída. O OpenMP divide as iterações entre os processadores disponíveis, fazendo com que num mesmo código existam trechos que rodam de forma seqüencial e trechos que rodam de forma paralela. Embora não seja um método que possibilite uma grande otimização do processamento, se comparado com os outros métodos de paralelismo, o OpenMP possui a vantagem de ser de fácil utilização, pois o processo de paralelização do código existente é extremamente simples.

PVM (Parallel Virtual Machine)

A idéia do PVM consiste em montar uma máquina virtual de *n* processadores e usá-los para executar tarefas simultâneas. O PVM é dividido em três partes principais:

- Console: usado para montar a máquina paralela virtual.
- Daemon: um programa que roda em cada máquina do ambiente estabelecendo a comunicação entre as diversas máquinas.

- Biblioteca: é na biblioteca que residem as funções e sub-rotinas que instruem o código a dividir as tarefas entre os daemons.

A biblioteca dispõe de recursos que possibilitam manipular qualquer elemento do seu ambiente virtual, inclusive em tempo de execução, embora não seja muito eficiente fazê-lo dessa forma, devido ao custo computacional de se adicionar e retirar máquinas. O ideal é criar a máquina virtual fora do código, através do console, e usá-la várias vezes, ou mesmo deixá-la ativa enquanto as máquinas estiverem ligadas, além de possibilitar disparar e matar processos a partir do console.

MPI (Message Passing Interface)

O MPI é uma tentativa de padronização do paradigma da troca de mensagens, que foi sugerida por um grupo de trabalho formado por pessoas da indústria, governo e universidades. Ele é constituído por um padrão de troca de mensagens com sintaxe definida, mas preservando características exclusivas de cada arquitetura, inclusive para arquiteturas de memória compartilhada. O principal objetivo do MPI é otimizar a comunicação e aumentar o desempenho computacional das máquinas, não possuindo dessa forma gerenciamento dinâmico de processos e processadores.

Embora exista a restrição citada acima, os programas escritos em MPI tendem a ser mais eficientes pelo fato de não haver acúmulo na carga de processos em tempo de execução. A diferença básica entre o MPI e o PVM é que, ao contrario do anterior, no MPI existe um único código fonte igual para todas as máquinas e conseqüentemente um único processo rodando.

Esta biblioteca, por proporcionar o desenvolvimento de códigos mais eficientes e permitir grande controle sobre a forma como os processadores se comunicam, foi adotada na elaboração dos algoritmos paralelos neste trabalho.

Assim, consideramos apropriado detalhar um pouco melhor seu funcionamento a seguir.

O MPI é uma biblioteca com funções para troca de mensagens, responsável pela comunicação e sincronização de processos. Dessa forma, os processos de um programa paralelo podem ser escritos em uma linguagem de programação seqüencial, tal como C ou Fortran.

O MPI funciona da seguinte forma: cada máquina ou nó recebe uma copia do código fonte e um nome. Cada nó começa a executar o programa a partir da primeira linha de comando utilizando as seguintes diretrizes:

- Executar todas as linhas de comando não nomeadas;
- Executar as linhas de comando nomeadas com o mesmo nome do nó;
- Não executar as linhas de comando com o nome de outro nó.

Para que o programa siga essas diretrizes, o procedimento padrão consiste na inclusão de vários comandos IF, com a seguinte estrutura: "Se eu sou o nó tal, faço isso... Senão faço aquilo...".

A programação em MPI utiliza um conjunto próprio de funções básicas de comunicação que iremos detalhar um pouco melhor, visando tornar mais simples o entendimento dos algoritmos paralelos propostos neste trabalho.

- **MPI_Init:** inicializa um processo MPI.

Sintaxe: int MPI_Init (int *argc, char *argv[ ])

Onde: argc - apontador para a quantidade de parâmetros da linha de comando

argv - apontador para um vetor de strings

- **MPI_COMM_RANK:** identifica um processo dentro de um determinado grupo.Retorna sempre um valor inteiro entre 0 e n-1, onde n é o número de processos.

Sintaxe: MPI_COMM_RANK (comm, rank);

Onde: comm - comunicador do MPI
rank - variável inteira com o numero de identificação do processo

- **MPI_COMM_SIZE:** retorna o número de processos dentro de um grupo.

Sintaxe: MPI_Comm_size (comm, size);

Onde: comm - comunicador do MPI
size - variável inteira que retorna o número de processos iniciados pelo MPI

- **MPI_Send:** rotina básica para envio de mensagens no MPI.

Sintaxe: MPI_Send (sndbuf, count, dtype, dest, tag, comm);

Onde: sndbuf - identificação do buffer (endereço inicial de onde os dados serão enviados)
count - número de elementos a serem enviados
dtype - tipo de dado
dest - identificação do processo destino
tag - rótulo (label) da mensagem
comm - comunicador do MPI

- **MPI_Recv:** rotina básica para recepção de mensagens no MPI.

Sintaxe: MPI_Recv (recvbuf, count, dtype, source, tag, comm, status);

Onde: recvbuf - identificação do buffer (endereço onde os dados serão recebidos)
count - número de elementos a serem recebidos
dtype - tipo de dado
source - identificação do processo emissor
tag - rótulo (label) da mensagem
comm - comunicador do MPI
status - vetor de informações envolvendo os parâmetros source e tag

- **MPI_Finalize:** finaliza um processo MPI. Portanto deve ser a última rotina a ser chamada por cada processo. Sincroniza todos os processos na finalização de uma aplicação MPI.

Sintaxe: MPI_Finalize();

Para obter maiores detalhes sobre o MPI consulte o site: www.lam-mpi.org. Lá estão disponíveis para download todos os softwares necessários para se trabalhar com o MPI, além de guias detalhados e cursos on-line sobre a ferramenta.

ANEXO B – O Cluster

O Cluster no qual todos os experimentos com algoritmos paralelos foram executados faz parte de um grande projeto conhecido com Tanque de Provas Numérico (TPN). Este projeto, conduzido em conjunto por uma série de universidades, empresas e institutos de pesquisa, com sede na Escola Politécnica da USP tem como objetivo criar um simulador de plataformas de petróleo e sistemas flutuantes para ser utilizado em conjunto com o tanque de provas físico no projeto de plataformas e embarcações.

Dada a grande complexidade deste tipo de análises que envolvem uma série de cálculos hidrodinâmicos e de elementos finitos surgiu a necessidade de se utilizar sistemas com alta capacidade de processamento. Neste sentido foi proposta a construção de um Cluster que oferecesse alto desempenho a um custo muito mais baixo do que a aquisição de um supercomputador equivalente.

O projeto do Cluster foi estruturado em três etapas: a construção de um protótipo com 10 nós, a criação do primeiro cluster com 60 nós do modelo Pentium III 866MHz e finalmente o segundo cluster com mais 60 nós Pentium IV 2.4GHz. Os experimentos relatados neste trabalho foram realizados no segundo cluster que será mais bem detalhado a seguir.

A especificação para a montagem de 63 máquinas, sendo 60 nós do cluster e mais 3 máquinas reserva foram as seguintes:

- Motherboard Intel GERG2LK
- Placa de rede Gigabit Ethernet on board
- Processador Intel Pentium 4 2.4 GHz com barramento de 533 MHz
- Memória DDR 333 de 512 Mbytes

- Hard Disk ATA 133 de 40 Gbytes
- Gabinete de 4U

A Figura 1.1.1 a seguir mostra a vista frontal parcial do cluster que foi montado com gabinetes tipo rack em armários com ventilação e sistema de suprimento de energia independentes. A Figura 1.1.2 apresenta a vista traseira do Cluster enfatizando o cabeamento de rede e o suprimento de energia garantido por um no-brake que permite o funcionamento das instalações por até 2h caso falte energia, localizado na parte inferior dos armários.

Figura A.1 - Vista frontal parcial do Cluster

Figura A.2 - Vista traseira parcial do Cluster

A comunicação entre os nós deste segundo Cluster é feita através de uma rede do tipo Gigabit Ethernet que permite um tráfego de rede 10 vezes maior do que as redes convencionais. Os três switches que controlam esta rede, o servidor de backup que armazena os dados dos casos rodados no cluster, o no-break e o nó de controle, que monitora a temperatura e as condições de operação dos demais nós podem ser vistos na figura A.3.

Figura A.3 - De cima para baixo: nó de controle, switches Gigabit (pretos), switches 100Mbit (brancos), servidor de backup e no-breake.

ANEXO C – Resultados Detalhados

A tabela abaixo apresenta o resultado detalhado da solução gerada pelo método do Algoritmo Genético Paralelo para o caso base da 4ª feira.

Rota	Lojas	Veículo	Tempo (h)	Distância (Km)	Frete (R$)
1	173, 193, 46	Truck	6,7	49,2	170,00
2	56, 139, 174	Leve	8,0	80,2	122,00
3	108, 43, 161	Truck	8,0	85,5	170,00
4	13, 211, 114	Truck	8,0	83,5	196,00
5	133, 130, 39	Truck	9,8	192,9	340,20
6	64, 155	Truck	9,1	218,7	267,80
7	132, 12	Truck	7,8	192,4	329,20
8	127, 40	Leve	8,7	201,1	222,20
9	27, 196	Truck	4,5	31,5	140,00
10	116, 20	Truck	9,0	56,2	154,00
11	201, 169	Truck	9,5	81,9	180,00
12	41, 34	Truck	8,8	206,3	283,20
13	192, 81	Carreta	8,9	42,3	206,00
14	11, 122	Leve	9,1	114,2	140,00
15	67, 48	Truck	5,6	58,2	154,00
16	24, 26	Truck	4,9	41,8	154,00
17	89, 90	Carreta	8,3	179,9	456,00
18	97, 101	Carreta	8,5	198,3	426,00
19	104, 74	Carreta	6,0	71,2	206,00
20	197, 22	Truck	5,8	67,1	154,00
21	137, 129	Truck	7,2	153,9	288,80
22	189, 92	Truck	7,2	52,9	154,00
23	47, 85	Truck	5,7	63,7	154,00
24	162, 163	Truck	9,6	106,2	180,00
25	148, 134	Truck	10,0	272,9	446,00
26	33, 203	Truck	9,5	76,5	154,00
27	65, 179	Truck	9,2	229,7	267,80
28	117, 55	Truck	9,4	49,5	154,00
29	115, 208	Truck	9,9	71,6	154,00
30	19, 213	Truck	5,4	54,2	154,00
31	66, 32	Truck	9,6	65,3	154,00
32	155, 118	Truck	6,5	99,2	180,00
33	57, 14	Truck	4,9	43,1	154,00
34	142, 171	Truck	5,4	58,3	154,00
35	129, 128	Truck	8,5	188,4	300,80
36	156, 54	Truck	9,2	78,3	154,00
37	153, 154	Truck	7,2	153,3	209,40
38	204, 121	Truck	9,6	70,5	154,00
39	38, 131	Truck	9,2	223,4	325,00

Rota	Lojas	Veículo	Tempo (h)	Distância (Km)	Frete (R$)
40	176, 45	Truck	7,4	61,2	154,00
41	190, 3	Truck	9,4	56,2	154,00
42	191, 143	Truck	9,5	63,2	154,00
43	158, 151	Truck	9,6	69,7	154,00
44	109, 135	Leve	7,8	154,1	211,20
45	5, 182	Truck	8,8	48,8	154,00
46	70, 120	Truck	9,9	275,9	271,20
47	124, 195	Truck	9,1	67,6	154,00
48	126, 138	Truck	8,9	208,4	213,00
49	149, 152	Truck	9,2	221,1	288,80
50	82, 83	Truck	8,1	95,9	154,00
51	25, 207	Truck	9,2	47,4	154,00
52	50, 210	Truck	8,9	227,4	271,20
53	141, 29	Truck	6,0	74,2	154,00
54	18, 140	Truck	9,8	78,4	154,00
55	113, 112	Truck	5,9	71,1	154,00
56	88, 202	Truck	8,4	185,4	283,20
57	119, 181	Leve	6,1	84,6	124,00
58	79, 172	Leve	8,3	82,5	124,00
59	17, 145	Truck	8,8	204,0	300,80
60	86, 157	Leve	7,0	108,9	106,00
61	160, 68	Truck	10,0	110,0	154,00
62	184, 194	Leve	8,1	165,4	154,60
63	23, 206	Leve	5,1	48,9	106,00
64	28, 111	Truck	5,2	48,6	154,00
65	185, 209	Truck	10,0	284,6	289,80
66	105, 87	Carreta	8,9	190,2	368,00
67	200, 157	Leve	6,8	99,6	106,00
68	198, 51	Truck	5,9	70,4	154,00
69	180, 144	Truck	9,1	50,7	154,00
70	1, 188	Truck	4,6	34,5	140,00
71	10, 178	Truck	7,6	169,7	300,80
72	44, 8	Truck	15,2	697,6	824,00
73	199, 42	Truck	5,8	66,2	154,00
74	213, 34	Truck	5,9	71,0	180,00
75	125, 214	Truck	9,8	76,9	154,00
76	168, 170	Truck	7,8	192,4	329,20
77	98	Carreta	5,9	165,9	433,00
78	93	Truck	4,3	79,0	164,00
79	80	Carreta	3,9	63,0	222,00
80	77	Carreta	6,2	182,7	392,00
81	69	Leve	3,1	36,6	90,00
82	187	Leve	2,6	22,3	82,00
83	166	Truck	2,2	13,0	124,00
84	84	Truck	4,4	84,4	138,00
85	60	Leve	7,7	294,3	227,20

Rota	Lojas	Veículo	Tempo (h)	Distância (Km)	Frete (R$)
86	189	Leve	2,8	26,4	90,00
87	103	Carreta	2,7	26,0	173,00
88	21	Leve	2,4	17,7	82,00
89	79	Carreta	4,2	74,7	222,00
90	122	Truck	5,1	114,1	172,00
91	61	Leve	9,7	444,4	338,40
92	73	Carreta	2,5	19,8	173,00
93	167	Truck	2,4	18,0	124,00
94	184	Truck	3,8	58,2	138,00
95	153	Leve	5,7	153,3	138,60
96	80	Carreta	3,9	63,0	222,00
97	146	Leve	7,9	311,4	401,20
98	53	Leve	3,4	45,1	90,00
99	94	Carreta	3,2	39,8	190,00
100	85	Leve	3,9	62,2	90,00
101	72	Carreta	6,2	182,7	392,00
102	37	Leve	13,7	697,6	573,00
103	86	Truck	4,2	74,0	138,00
104	183	Leve	8,4	347,6	249,20
105	31	Leve	9,7	444,4	338,40
106	15	Truck	2,9	30,8	124,00
107	150	Leve	2,4	18,0	82,00
108	4	Leve	3,2	37,0	90,00
109	99	Truck	12,1	602,7	684,00
110	118	Truck	4,5	86,8	164,00
111	91	Carreta	3,4	43,9	190,00
112	71	Truck	3,2	39,7	138,00
113	58	Truck	8,9	380,7	373,80
114	30	Leve	8,4	347,6	249,20
115	52	Leve	3,2	39,3	90,00
116	123	Truck	4,2	74,2	164,00
117	76	Carreta	3,6	51,3	190,00
118	75	Carreta	3,9	61,0	190,00
119	102	Truck	2,7	24,2	124,00
120	82	Truck	4,0	64,6	138,00
121	78	Truck	10,9	525,7	607,00
122	106	Leve	2,4	16,7	82,00
123	152	Truck	5,7	153,3	193,40
124	75	Carreta	3,9	61,0	190,00
125	98	Leve	5,9	165,9	203,20
126	35	Truck	2,8	27,4	138,00
127	76	Carreta	3,6	51,3	190,00
128	159	Truck	3,7	54,9	164,00
129	205	Leve	3,1	36,6	90,00
130	100	Carreta	4,6	92,3	222,00
131	36	Leve	4,5	88,5	130,60

Rota	Lojas	Veículo	Tempo (h)	Distância (Km)	Frete (R$)
132	174	Truck	3,3	42,1	138,00
133	101	Carreta	3,2	37,2	190,00
134	62	Truck	9,7	444,4	475,60
135	122	Truck	5,1	114,1	172,00
136	17	Truck	5,7	153,3	193,40
137	84	Truck	4,4	84,4	138,00
138	72	Leve	6,2	182,7	180,80
139	98	Carreta	5,9	165,9	433,00
140	101	Carreta	3,2	37,2	190,00
141	165	Leve	4,5	86,8	108,00
142	72	Carreta	6,2	182,7	392,00
143	79	Carreta	4,2	74,7	222,00
144	175	Leve	6,3	192,4	222,80
145	177	Truck	2,0	10,1	124,00
146	107	Truck	3,8	59,2	164,00
147	101	Carreta	3,2	37,2	190,00
148	71	Truck	3,2	39,7	138,00
149	110	Truck	3,5	48,9	138,00
150	55	Truck	3,2	37,1	138,00
151	101	Carreta	3,2	37,2	190,00
152	59	Truck	9,0	389,8	464,80
153	136	Leve	6,3	192,4	222,80
154	134	Truck	6,9	234,3	430,00
155	9	Leve	6,2	182,7	180,80
156	147	Truck	8,3	335,4	494,80
157	2	Truck	3,0	32,1	138,00
158	212	Truck	17,2	914,1	1.097,20
159	96	Carreta	5,7	153,3	294,00
160	59	Leve	9,0	389,8	333,20
161	63	Leve	10,3	485,2	360,40
162	154	Truck	5,7	153,3	193,40
163	164	Truck	4,3	77,7	164,00
164	103	Carreta	2,7	26,0	173,00
165	92	Leve	3,6	52,6	90,00
166	92	Leve	3,6	52,6	90,00
167	92	Truck	3,6	52,6	138,00
168	49	Truck	2,8	26,4	124,00
169	16	Leve	3,9	62,5	90,00
170	6	Leve	3,3	41,2	90,00
171	186	Truck	3,0	32,3	138,00
172	7	Leve	8,4	347,6	249,20
173	95	Truck	5,7	153,3	193,40
174	92	Truck	3,6	52,6	138,00

Tabela A.1 - Resultados detalhados (elaborado pelo autor)

ANEXO D – Código Fonte dos Programas

Algoritmos genéticos - a versão apresentada abaixo é a paralela. Caso se deseje a seqüencial, podemos considerar este programa executado em apenas um computador:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

#define NMAX 260        //maximo de numero de lojas
#define TIPO 3          //tipos de caminhao
#define POP 60          //Numero de elementos na populacao
#define TAM 4           //Tamanho maximo de cada rota
#define MAXROT 550      //Numero maximo de rotas possiveis
#define FATOR 1.27      //Fator de correcao das distancias
#define CUST 5          //Componentes de custo
#define VMAX 10         //maximo de visitas a uma mesma loja
#define MPITAG 1


double T = 10.0;                //jornada do motorista em h
double V = 175.0;               //velocidade media em km/h
double Temp = 1.5;              //tempo medio de parada em h

double falt = 1.0, jorn = 1.0, jan = 1.0;//penalidades

//parametros do algoritimo

int GER = 10000;                //Numero de geracoes
double PMUT = 0.0001;           //Probabilidade de ocorrer mutacao
double PTROCA = 0.0001;         //Probabilidade de ocorrer troca
double PCROSS = 0.7;            //Probabilidade de ocorrer crossover
int MRANGE = 6;         //Amplitude da mutacao
int BEST = 5;           //Numero de elementos a serem armazenados por geracao
int FREQ = 3;           //intervalo de geracoes em que ocorre migracao


int N;                                  //Numero de lojas
int ROT;                                //Numero maximo de rotas
double hi[NMAX+1], hf[NMAX+1];          //janela de recebimento da loja i
double dist[NMAX+1][NMAX+1];            //distacia entre duas lojas
double tempop[NMAX+1][NMAX+1];          //tempo de percurso entre as lojas
double X[NMAX+1], Y[NMAX+1];            //localizacao da loja
double custofaixa[NMAX+1][TIPO+1];      //custo para atender a loja
double delta[NMAX+1][TIPO+1];           //custo para atender uma loja adicional na mesma rota
int carreta[NMAX+1];                    //loja pode receber carreta
double d[NMAX+1];                       //demanda
double cap[TIPO+1];                     //capacidade dos veiculos
int pop0[POP+1][MAXROT+1][TAM+1];  //solucoes
```

```
double pop1[POP+1][MAXROT+1][TAM+1];      //solucoes
int store0[POP+1][MAXROT+1][TAM+1];       //armazena solucoes
double store1[POP+1][MAXROT+1][TAM+1];    //armazena solucoes


double custo[POP+1][CUST];                /armazena todas as categorias de custo
double distrota[POP+1][MAXROT+1];         //armazena a distancia de cada rota
double tempo[POP+1][MAXROT+1];            //armazena a duracao de cada rota
double hora[POP+1][NMAX+1][VMAX+1];       //armazena o horario em que cada loja foi
atendida
double entrega[POP+1][NMAX+1];            //armazena a quantidade entregue para cada
loja



//Funções

int rnd(double min, double max)           //Gera um numero inteiro aleatorio entre min e max
{
   int x;
   double range = 0.0;
   range = (max - min + 1);
   x = min + (int) (range*rand() / (RAND_MAX+1.0));
   return x;
}

double randomico(double n)
{
        double x;

        x = (n * rand() / (RAND_MAX+1.0));

        return x;
}

int mod(int x)
{
        if(x >= 0)
                return x;
        return -x;
}

double modulo(double x)
{
        if(x >= 0)
                return x;
        return -x;
}

double maximo(double a, double b)
{
        if(a > b)
                return a;
        return b;
}
```

```
void calcdist()
{
	int i, j;

	for(i = 0; i <= N; i++)
		for(j = 0; j <= N; j++)
			dist[i][j] = 6377*(acos(sin(X[i])*sin(X[j]) +
		cos(X[i])*cos(X[j])*cos(modulo(Y[j]-Y[i]))))*FATOR;
}
double veloc(double x)
{
	double vel;

	vel = 16.952 + 0.3096*x + (-0.0008)*pow(x,2) + 0.0000007*pow(x,3);
	if(vel > 60)
		vel = 60;

	return vel;
}

void calctempop()
{
	int i, j;

	for(i = 0; i <= N; i++)
		for(j = 0; j <= N; j++)
			tempop[i][j] = dist[i][j] / veloc(dist[i][j]);

}

void calcrota()
{
    int i, j, k;

    for(i = 1; i <= POP; i++)
		for(j = 1; j <= ROT; j++)
			distrota[i][j] = 0.0;

    for(i = 1; i <= POP; i++)
		for(j = 1; j <= ROT; j++)
		{
			distrota[i][j] = distrota[i][j] + dist[0][pop0[i][j][1]] +  dist[pop0[i][j][TAM]][0];
			for(k = 1; k < TAM; k++)
				distrota[i][j] = distrota[i][j] + dist[pop0[i][j][k]][pop0[i][j][k+1]];
		}
}

void gerar()
{
	int i, j, k;

	for(i = 1; i <= POP; i++)
		for(j = 1; j <= ROT; j++)
		{
```

```
                        pop0[i][j][0] = rnd(1,TIPO);
                        for(k = 1; k <= TAM; k++)
                        {
                                pop0[i][j][k] = rnd(1,N);
                                pop1[i][j][k] = 0;
                        }
                }
}

void arrumar()
{
        int i, j, k, l, soma[MAXROT+1], ent[NMAX+1];

        for(i = 1; i <= POP; i++)
        {
                for(j = 1; j <= ROT; j++) //remove as lojas que nao recebem carreta. da rota feita
por carr.
                        for(k = 1; k <= TAM; k++)
                                if(pop0[i][j][0] == 1 && carreta[pop0[i][j][k]] == 0)
                                        {
                                                pop0[i][j][k] = 0;
                                                pop1[i][j][k] = 0;
                                        }

                for(j = 1; j <= ROT; j++)
                        for(k = 1; k <= TAM; k++)
                        {
                                if(pop1[i][j][k] == 0) //retira da rota as lojas onde a entrega eh 0
                                        pop0[i][j][k] = 0;
                                if(pop0[i][j][k] == 0) //retira o volume entregue para loja 0
                                        pop1[i][j][k] = 0;
                        }

                for(j = 1; j <= ROT; j++)  // retira os zeros das rotas
                        for(l = 1; l < TAM; l++)
                                for(k = 1; k < TAM; k++)
                                        if(pop0[i][j][k] == 0)
                                        {
                                                pop0[i][j][k] = pop0[i][j][k+1];
                                                pop1[i][j][k] = pop1[i][j][k+1];
                                                pop0[i][j][k+1] = 0;
                                                pop1[i][j][k+1] = 0;
                                        }
        }
}

void atribui()
{
   int i, j, k, l;
        double capac[MAXROT+1], ent[NMAX+1];
   double dif;

        for(i = 1; i <= POP; i++)
        {
     for(j = 1; j <= ROT; j++)
```

```
            capac[j] = cap[pop0[i][j][0]];
        for(j = 1; j <= N; j++)
            ent[j] = 0;
        for(j = 1; j <= ROT; j ++)
            for(k = 1; k <= TAM; k++)
            {
                dif = d[pop0[i][j][k]] - ent[pop0[i][j][k]];
                if(dif > 0)
                {
                    if(capac[j] >= dif)
                    {
                        pop1[i][j][k] =  pop1[i][j][k] + dif;
                        ent[pop0[i][j][k]] = ent[pop0[i][j][k]] + dif;
                        capac[j] = capac[j] - dif;
                    }
                    else
                    {
                        pop1[i][j][k] =  pop1[i][j][k] + capac[j];
                        ent[pop0[i][j][k]] = ent[pop0[i][j][k]] + capac[j];
                        capac[j] = 0;
                    }
                }
            }
    }
}
void calcentrega()
{
            int i, j, k;

            for(i = 1; i <= POP; i++)
                        for(j = 1; j <= N; j++)
                                    entrega[i][j] = 0;

            for(i = 1; i <= POP; i++)
                        for(j = 1; j <= ROT; j++)
                                    for(k = 1; k <= TAM; k++)
                                                entrega[i][pop0[i][j][k]] = entrega[i][pop0[i][j][k]] + pop1[i][j][k];
}

void calctempo() //calcula o tempo total da rota
{
            int i, j, k;

            for(i = 1; i <= POP; i++)
                        for(j = 1; j <= ROT; j++)
                                    tempo[i][j] = 0.0;

            for(i = 1; i <= POP; i++)
                        for(j = 1; j <= ROT; j++)
                        {
                                    for(k = 1; k < TAM; k++)   //tempo de percurso
                                                tempo[i][j] = tempo[i][j] + tempop[pop0[i][j][k]][pop0[i][j][k+1]];
                                    tempo[i][j] = tempo[i][j] + tempop[0][pop0[i][j][1]] +
tempop[pop0[i][j][TAM]][0];
```

```
                        for(k = 1; k <= TAM; k++)  //tempo de parada
                                if(pop0[i][j][k] > 0)
                                        tempo[i][j] = tempo[i][j] + Temp;
                }

}

void calchora()     //calcula a hora em que cada loja é atendida e o
{               //tempo total da rota
        int i, j, k, l, n;
        double t;

        for(i = 1; i <= POP; i++)
                for(j = 1; j <= N; j++)
        for(k = 1; k <= VMAX; k++)
                hora[i][j][k] = 0.0;

        for(i = 1; i <= POP; i++)
                for(j = 1; j <= ROT; j++)
                        tempo[i][j] = 0.0;

        for(i = 1; i <= POP; i++)
                for(j = 1; j <= ROT; j++)
                {
                        if(pop0[i][j][1] > 0)
                        {
                                n = 1;
                                t = hi[pop0[i][j][1]];
                                for(l = 1; ((hora[i][pop0[i][j][1]][l] > 0)&&(l <= VMAX)); l++)
                                        l=l;
                                hora[i][pop0[i][j][1]][l] = t;
                                tempo[i][j] = tempop[0][pop0[i][j][1]];
                                for(k = 2; k <= TAM; k++)
                                {
                                        for(l = 1; ((hora[i][pop0[i][j][k]][l] > 0)&&(l <= VMAX)); l++)
                                                l=l;
                                        if(pop0[i][j][k] > 0)
                                        {
                                                t = t + Temp + tempop[pop0[i][j][k-
1]][pop0[i][j][k]];
                                                tempo[i][j] = tempo[i][j] + Temp +
tempop[pop0[i][j][k-1]][pop0[i][j][k]];
                                                if(t < hi[pop0[i][j][k]])
                                                {
                                                        tempo[i][j] = tempo[i][j] + hi[pop0[i][j][k]] -
t;
                                                        t = hi[pop0[i][j][k]];
                                                }
                                                hora[i][pop0[i][j][k]][l] = t;
                                                n++;
                                        }

                                }

                                tempo[i][j] = tempo[i][j] + Temp + tempop[pop0[i][j][n]][0];
```

```
                        }
                }
}

void calccusto()
{
        int i, j, k, n;
        double cf, cd, e;

        calcentrega();
        calcrota();
        calchora();


        for(i = 0; i <= POP; i++)
                for(j = 0; j <= CUST; j++)
                        custo[i][j] = 0.0;

        for(i = 1; i <= POP; i++)//penaliza ultrapassar a jornada maxima do motorista
                for(j = 1; j <= ROT; j++)
                        if(tempo[i][j] > T)
                                custo[i][3] = custo[i][3] + (tempo[i][j] - T)*500*jorn;

        for(i = 1; i <= POP; i++)//penaliza ultrapassar a janela de recebimento
                for(j = 1; j <= N; j++)
        for(k = 1; k <= VMAX; k++)
                if(hora[i][j][k] > hf[j])
                        custo[i][4] = custo[i][4] + (hora[i][j][k] - hf[j])*5000*jan;


        for(i = 1; i <= POP; i++) //custo de falta e sobra
                for(j = 1; j <= N; j++)
                        custo[i][2] = custo[i][2] + modulo(d[j] - entrega[i][j])*5000*falt;


        for(i = 1; i <= POP; i++) //custo de transporte
                for(j = 1; j <= ROT; j++)
                {
                        cf = 0.0;
                        cd = 0.0;
                        n = 0;
                        for(k = 1; k <= TAM; k++)
                        {
                                if(custofaixa[pop0[i][j][k]][pop0[i][j][0]] > cf)
                                {
                                        cf = custofaixa[pop0[i][j][k]][pop0[i][j][0]];
                                        cd = delta[pop0[i][j][k]][pop0[i][j][0]];
                                }
                                if(pop0[i][j][k] != 0)
                                        n++;
                        }
                        custo[i][1] = custo[i][1] + cf + (n-1) * cd;
                        //printf("%lf %lf %d\n", cf, cd, n);
                }
```

```
        for(i = 1; i <= POP; i++) // custo total
             for(j = 1; j <= CUST; j++)
                  custo[i][0] = custo[i][0] + custo[i][j];

}

void rfalt()
{
        int i, j, k, stop, tipoc;

        for(i = 1; i <= POP; i++)
                if(custo[i][2] > 0)
                        for(j = 1; j <= N; j++)
                                if(d[j] > entrega[i][j])
                                {
                                        if(carreta[j] = 1)
                                                tipoc = 1;
                                        else
                                                tipoc = 2;
                                        stop = 0;
                                        for(k = 1; (k <= ROT )&&(stop = 0); k++)
                                                if(pop0[i][k][1] == 0)
                                                {
                                                        pop0[i][k][1] = j;
                                                        pop0[i][k][0] = tipoc;
                                                        stop = 1;
                                                }
                                }
}

void evoluir2()
{
        int i, j, k, x, y, z, num;
        double min;

        for(i = 1; i <= BEST; i++) //Seleciona e armazena as 'BEST' melhores soluÃ§Ãµes da
populaÃ§Ã£o anterior
        {
                min = 99999999;
                for(j = 1; j <= POP; j++)
                   if(custo[j][0] < min)
                {
                                min = custo[j][0];
                num = j;
                }
                custo[num][0] = 99999999;
                for(j = 1; j <= ROT ; j++)
                        for(k = 0; k <= TAM; k++)
                                        store0[i][j][k] = pop0[num][j][k];

        }

        for(i = 1; i <= BEST; i++) //copia de volta na populacao
                for(j = 1; j <= ROT ; j++)
                        for(k = 0; k <= TAM; k++)
```

```
                        pop0[i][j][k] = store0[i][j][k];

for(i = BEST + 1; i <= POP; i++)
{
        x = rnd(1, BEST);
        y = rnd(1, BEST);
        z = rnd(1, ROT);
        for(k = 0; k <= TAM; k++)
        {
                for(j = 1; j < z; j++)
                {
                        pop0[i][j][k] = store0[x][j][k];
                        pop0[i+1][j][k] = store0[y][j][k];
                }
                for(j = z; j <= ROT; j++)
                {
                        pop0[i][j][k] = store0[y][j][k];
                        pop0[i+1][j][k] = store0[x][j][k];
                }
        }

        for(i = 2; i <= POP; i++) // mutações
                for(j = 1; j <= ROT; j++)
                {
                        for(k = 1; k <= TAM; k++)
                                if(randomico(1) <= PMUT)
                                        pop0[i][j][k] = rnd(0,N); //na loja

                        if(randomico(1) <= PMUT)
                                pop0[i][j][0] = rnd(1,TIPO);// no tipo do caminhao
                }
        for(i = 2; i <= POP; i++) // troca entre rotas
                for(j = 1; j <= ROT; j++)
                        for(k = 1; k <= TAM; k++)
                                if(randomico(1) <= PTROCA)
                                {
                                        z = rnd(2, POP);
                                        y = rnd(1, TAM);
                                        x = pop0[i][z][y];
                                        pop0[i][z][y] = pop0[i][j][k];
                                        pop0[i][j][k] = x;
                                }
        for(i = 2; i <= POP; i++) // troca na rota
                for(j = 1; j <= ROT; j++)
                        for(k = 1; k <= TAM; k++)
                                if(randomico(1) <= PTROCA)
                                {
                                        z = rnd(1, TAM);
                                        x = pop0[i][j][z];
                                        pop0[i][j][z] = pop0[i][j][k];
                                        pop0[i][j][k] = x;
                                }
}
for(i = 1; i <= POP; i++)
        for(j = 1; j <= ROT; j++)
```

```
                        for(k = 1; k <= TAM; k++)
                                pop1[i][j][k] = 0;

}
void saida(int num)
{
        int i, j, p, n;

        FILE *arqsaida;
        char nome[100];

        sprintf(nome, "/home/winckler/guilherme2/saida/saida2.txt"); // dados das lojas
        arqsaida = fopen(nome,"w");

        for(i = 1; i <= ROT; i++)
        {
                if(pop0[num][i][1] > 0)
                {
                        p = 0;
                        n = 1;
                        fprintf(arqsaida, "0\n");
                        for(j = 1; j <= TAM; j++)
                        {
                                if(p == 0)
                                {
                                        fprintf(arqsaida, "%d\n", pop0[num][i][j]);
                                        if(pop0[num][i][j] > 0)
                                                p = j;
                                }
                                else
                                {
                                        if(pop0[num][i][j] > 0)
                                        {
                                                fprintf(arqsaida, "%d\n", pop0[num][i][j]);
                                                p = j;
                                        }
                                        else
                                                n++;
                                }
                        }
                        for(j = 1; j <= n; j++)
                        fprintf(arqsaida, "0\n");
                        fprintf(arqsaida, "\n");
                }
        }
        fclose(arqsaida);

        sprintf(nome, "/home/winckler/guilherme2/saida/cam2.txt"); // caminhoes utilizados em
cada rota
        arqsaida = fopen(nome,"w");
   for(i = 1; i <= ROT; i++)
                if(pop0[num][i][1] > 0)
                        fprintf(arqsaida, "%d\n\n\n\n\n\n\n\n", pop0[num][i][0]);
   fclose(arqsaida);
```

```
   sprintf(nome, "/home/winckler/guilherme2/saida/detrota2.txt"); // caminhoes utilizados em
cada rota
   arqsaida = fopen(nome,"w");
   for(i = 1; i <= ROT; i++)
                    if(pop0[num][i][1] > 0)
                              fprintf(arqsaida, "%d\t%lf\t%lf\n",i,  distrota[num][i], tempo[num][i]);

   fclose(arqsaida);

   sprintf(nome, "/home/winckler/guilherme2/saida/detloja2.txt");
   arqsaida = fopen(nome,"w");
   for(i = 1; i <= N; i++)
   {
        for(j = 1; j <= VMAX; j++)
                    fprintf(arqsaida, "%d\t%lf", i, hora[num][i][j]);
        fprintf(arqsaida, "\t%lf\n", entrega[num][i]);
   }

   fclose(arqsaida);

   sprintf(nome, "/home/winckler/guilherme2/saida/rotas2.txt");
   arqsaida = fopen(nome,"w");
   fprintf(arqsaida, "%lf\t%lf\t%lf\t%lf\t%lf\n", custo[num][0], custo[num][1], custo[num][2],
custo[num][3], custo[num][4]);
   for(i = 1; i <= ROT; i++)
   {
        if(pop0[num][i][1] > 0)
        {
             fprintf(arqsaida, "%d", pop0[num][i][0]);
             for(j = 1; pop0[num][i][j]; j++)
                       fprintf(arqsaida, "\t%d", pop0[num][i][j]);
             fprintf(arqsaida, "\n");
        }
   }
   fclose(arqsaida);
}

int main(int argc, char *argv[])
{
          int i, j, k, l, g, num, n, count;
          int rank, size;
          double bestc, crot;

          FILE *arqentrada;
          char nome[100];
          MPI_Status status;

          printf("Numero ");
          scanf("%d", &num);
          srand(num);

          sprintf(nome, "/home/winckler/guilherme2/DadosTXT/Dados28-02.txt"); // dados das
lojas
          arqentrada = fopen(nome,"r");
          fscanf(arqentrada, "%d", &N);
```

```
for(i=0; i<= N; i++)
{
	fscanf(arqentrada, "%lf", &X[i]);
	fscanf(arqentrada, "%lf", &Y[i]);
	fscanf(arqentrada, "%lf", &d[i]);
	fscanf(arqentrada, "%lf", &custofaixa[i][1]);
	fscanf(arqentrada, "%lf", &custofaixa[i][2]);
	fscanf(arqentrada, "%lf", &custofaixa[i][3]);
	fscanf(arqentrada, "%lf", &delta[i][1]);
	fscanf(arqentrada, "%lf", &delta[i][2]);
	fscanf(arqentrada, "%lf", &delta[i][3]);
	fscanf(arqentrada, "%lf", &hi[i]);
	fscanf(arqentrada, "%lf", &hf[i]);
	fscanf(arqentrada, "%d", &carreta[i]);
}
fclose (arqentrada);

ROT = 0;
crot = 0.0;
for(i = 1; i <= N; i++)  //calcula o numero maximo de rotas permitido
crot = crot + d[i];
crot = (crot / 14) + N;
ROT = (int)crot;

MPI_Init(&argc, &argv);

MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

if(rank == 0)
{
	for(i = 1; i < size; i++)
	{
		n = rnd(1,1000);
		MPI_Send(&n, 1, MPI_INT, i, MPITAG, MPI_COMM_WORLD);
	}
}
else
{
	MPI_Recv(&num, 1, MPI_INT, 0, MPITAG, MPI_COMM_WORLD, &status);
	srand(num);
}

for(i = 0; i <= POP; i++)
	for(j = 0; j <= ROT; j++)
		for(k = 0; k <= TAM; k++)
		{
			pop0[i][j][k] = 0;
			pop1[i][j][k] = 0;
		}

for(i = 0; i <= POP; i++)
	for(j = 0; j <= ROT; j++)
		for(k = 0; k <= TAM; k++)
		{
```

```
                                store0[i][j][k] = 0;
                                store1[i][j][k] = 0;
                        }

        for(i = 0; i <= POP; i++)
                for(j = 0; j <= N; j++)
                        entrega[i][j] = 0;

        calcdist();
        calctempop();

        cap[1] = 28.0;
        cap[2] = 14.0;
        cap[3] = 7.0;

        bestc = 9999999999.0;
        num = 1;

        gerar();

        count = 1;
        for(g = 1; g <= GER; g++)
        {
                bestc = 9999999999.0;
                num = 1;

                if(g == 1)
                {
                        sprintf(nome, "/home/winckler/guilherme/AG.txt"); // carrega solucao
inicial
                        arqentrada = fopen(nome,"r");

                        for(i = 1; i <= ROT; i++)
                        {
                for(j = 0; j <= TAM; j++)
                        fscanf(arqentrada, "%d", &pop0[1][i][j]);

                for(j = 0; j <= TAM; j++)
                fscanf(arqentrada, "%lf", &pop1[1][i][j]);
                }

                        fclose (arqentrada);
                }

                for(i = 1; i <= ROT; i++)
                        for(j = 1; j <= TAM; j++)
                                pop1[1][i][j] = 0;

                atribui();
                arrumar();
                calccusto();

                j = 0;
                for(i = 1; i <= POP; i++)
                        if(custo[i][0] < bestc)
```

```
                        {
                                bestc = custo[i][0];
                                num = i;
                                j = 1;
                        }

                if((j == 1)&&(rank == 0))
                        printf("g: %d p: %d Total: %.6g Fret: %.5g Falt: %.5g Jorn: %.5g Jan:
%.5g\n", g, num, custo[num][0], custo[num][1], custo[num][2], custo[num][3], custo[num][4]);


                if(custo[num][2] < 40000)
                falt = custo[num][2] / 2 / 40000 + 0.5;
        else
                falt = 1.0;

        if(custo[num][3] < 20000)
                jorn = custo[num][3] / 2 / 20000 + 0.5;
        else
                jorn = 1.0;

                if(custo[num][4] < 20000)
                jan = custo[num][4] / 2 / 20000 + 0.5;
        else
                jan = 1.0;

                if(count == FREQ)
                {
                        if(rank < size - 1)
                                for(i = 1; i <= ROT; i++)
                                        for(j = 0; j <= TAM; j++)
                                        {
                                                MPI_Send(&pop0[num][i][j], 1, MPI_INT, rank +
1, MPITAG, MPI_COMM_WORLD);
                                                MPI_Send(&pop1[num][i][j], 1, MPI_INT, rank +
1, MPITAG, MPI_COMM_WORLD);
                                        }
                        else
                                for(i = 1; i <= ROT; i++)
                                        for(j = 0; j <= TAM; j++)
                                        {
                                                MPI_Send(&pop0[num][i][j], 1, MPI_INT, 0,
MPITAG, MPI_COMM_WORLD);
                                                MPI_Send(&pop1[num][i][j], 1, MPI_INT, 0,
MPITAG, MPI_COMM_WORLD);
                                        }
                        if(rank > 0)
                                for(i = 1; i <= ROT; i++)
                                        for(j = 0; j <= TAM; j++)
                                        {
                                                MPI_Recv(&pop0[POP][i][j], 1, MPI_INT, rank -
1, MPITAG, MPI_COMM_WORLD, &status);
                                                MPI_Recv(&pop1[POP][i][j], 1, MPI_INT, rank -
1, MPITAG, MPI_COMM_WORLD, &status);
                                        }
```

```
                        else
                                for(i = 1; i <= ROT; i++)
                                        for(j = 0; j <= TAM; j++)
                                        {
                                                MPI_Recv(&pop0[POP][i][j], 1, MPI_INT, size -
1, MPITAG, MPI_COMM_WORLD, &status);
                                                MPI_Recv(&pop1[POP][i][j], 1, MPI_INT, size -
1, MPITAG, MPI_COMM_WORLD, &status);
                                        }
                                count = 0;
                }

                if(g == GER)
                {
                        if(rank > 0)
                        {
                                for(i = 1; i <= ROT; i++)
                                        for(j = 0; j <= TAM; j++)
                                        {
                                                MPI_Send(&pop0[num][i][j], 1, MPI_INT, 0,
MPITAG, MPI_COMM_WORLD);
                                                MPI_Send(&pop1[num][i][j], 1, MPI_INT, 0,
MPITAG, MPI_COMM_WORLD);
                                        }
                        }
                        if(rank == 0)
                        {
                                for(i = 1; i <= ROT; i++)
                                        for(j = 0; j <= TAM; j++)
                                        {
                                                pop0[POP][i][j] = pop0[num][i][j];
                                                pop1[POP][i][j] = pop0[num][i][j];
                                        }
                                for(k = 1; k < size; k++)
                                        for(i = 1; i <= ROT; i++)
                                                for(j = 0; j <= TAM; j++)
                                                {
                                                        MPI_Recv(&pop0[k][i][j], 1, MPI_INT, k,
MPITAG, MPI_COMM_WORLD, &status);
                                                        MPI_Recv(&pop1[k][i][j], 1, MPI_INT, k,
MPITAG, MPI_COMM_WORLD, &status);
                                                }
                        }
                        if(rank > 0)
                        {
                                for(i = 0; i < CUST; i++)
                                        MPI_Send(&custo[num][i], 1, MPI_DOUBLE, 0,
MPITAG, MPI_COMM_WORLD);
                        }

                        if(rank == 0)
                        {
                                for(i = 0; i < CUST; i++)
                                        custo[POP][i] = custo[num][i];
                                for(j = 0; j < CUST; j++)
```

```
                                        for(i = 1; i < size; i++)
                                                MPI_Recv(&custo[i][j], 1, MPI_DOUBLE, i,
MPITAG, MPI_COMM_WORLD, &status);

                                bestc = 9999999999.0;

                                for(i = 1; i <= POP; i++)
                                        if(custo[i][0] < bestc)
                                        {
                                                bestc = custo[i][0];
                                                num = i;
                                        }

                                saida(num);
                                printf("%lf\n", custo[num][0]);
                        }

                }

                evoluir2();
                count++;
        }

        MPI_Finalize();
        return 0;
}
```

Clarke & Wright - o algoritmo a seguir está adaptado ao Meta-RaPS. Para torná-lo equivalente a heurística original, basta fazer o número de iterações igual a 1:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NMAX 260
#define TIPO 3
#define FATOR 1.27
#define TAM 4

int GER = 10000;                //número de gerações
double PH = 0.85;               //probabilidade de se seguir a regra heuristica
double T = 10.0;                //jornada do motorista em h
double Temp = 1.5;              //tempo médio de parada  em h

int N;                          //numero de lojas
double cap[TIPO+1];
int roteiro[NMAX+1][NMAX+2];
```

```
int listai[(NMAX*(NMAX-1))/2], listaj[(NMAX*(NMAX-1))/2];
int carreta[NMAX+1];                // loja aceita carreta
double horario[NMAX+1]; //horário no qual a loja i é atendida
double hi[NMAX+1], hf[NMAX+1];      //janela de recebimento da loja i
double listae[(NMAX*(NMAX-1))/2];
double d[NMAX+1];                   //demanda
double dem[NMAX+1];                 //demanda temporária
double bdem[NMAX+1];                //armazena a demanda temporaria da melhor solucao
double dist[NMAX+1][NMAX+1];        //distancia entrer lojas
double tempop[NMAX+1][NMAX+1];      //tempo de percurso entre as lojas
double X[NMAX+1], Y[NMAX+1];        //posição das lojas
double custofaixa[NMAX+1][TIPO+1], delta[NMAX+1][TIPO+1]; //custos de transporte
int camrota[NMAX+1];                //tipo de caminha que percorre a rota
double fretep[NMAX+1];              //frete pago por rota

int setup[NMAX+1][TIPO+1];          //entregas iniciais

int bcamrota[NMAX+1];
int broteiro[NMAX+1][NMAX+2];       //armazena melhores valores
int bsetup[NMAX+1][TIPO+1];
double bfretep[NMAX+1];

double randomico(double n)
{
        double x;

        x = (n * rand() / (RAND_MAX+1.0));

        return x;
}

int rnd(double min, double max) //Gera um numero inteiro aleatorio entre min e max
{
   int x;
   double range = 0.0;
   range = (max - min + 1);
   x = min + (int) (range*rand() / (RAND_MAX+1.0));
   return x;
}

double mod(double x)
{
        if(x >= 0)
                return x;
        return -x;
}

int contno(int r)
{
        int i;
        int cont = 0;

        for(i = 1; roteiro[r][i] > 0; i++)
                cont++;
```

```
	return cont;
}

void calcdistxy()
{
	int i, j;

	for(i = 0; i <= N; i++)
		for(j = 0; j <= N; j++)
			dist[i][j] = 6377*(acos(sin(X[i])*sin(X[j]) + cos(X[i])*cos(X[j])*cos(mod(Y[j]-
Y[i])))*FATOR;
}


double veloc(double x)
{
	double vel;

	vel = 16.952 + 0.3096*x + (-0.0008)*pow(x,2) + 0.0000007*pow(x,3);
	if(vel > 60)
		vel = 60;
	return vel;
}

void calctempop()
{
	int i, j;

	for(i = 0; i <= N; i++)
		for(j = 0; j <= N; j++)
			tempop[i][j] = dist[i][j] / veloc(dist[i][j]);

}

void calchorario(int r)
{
	int i;
	double tempo;

	tempo = hi[roteiro[r][1]];
	horario[roteiro[r][1]] = tempo;
	for(i = 2; roteiro[r][i] > 0; i++)
	{
		if(tempo + Temp  + tempop[roteiro[r][i-1]][roteiro[r][i]] >= hi[roteiro[r][i]])
			tempo = tempo + Temp + tempop[roteiro[r][i-1]][roteiro[r][i]];
		else
			tempo = hi[roteiro[r][i]];
		horario[roteiro[r][i]] = tempo;
	}
}

void entregasini()
{
	int i, j, k;
```

```
	for(i = 1; i <= N; i++)
		for(j = 1; j <= TIPO; j++)
			setup[i][j] = 0;
	for(i = 1; i <= N; i++) //para cada loja
	{
		if(carreta[i] == 1) //se a loja aceita carreta
		{
			while(dem[i] > cap[1])
			{
				setup[i][1] = setup[i][1] + 1;
				dem[i] = dem[i] - cap[1];
			}
		}
		else //se nao aceita carreta
		{
			while(dem[i] > cap[2])
			{
				setup[i][2] = setup[i][2] + 1;
				dem[i] = dem[i] - cap[2];
			}
		}
	}
}

void entregasini2()
{
	int i, j, k, x;

	for(i = 1; i <= N; i++)
		for(j = 1; j <= TIPO; j++)
			setup[i][j] = 0;

	for(i = 1; i <= N; i++) //para cada loja
	{
		if(carreta[i] == 1) //se a loja aceita carreta
		{
			while(dem[i] > cap[1])
			{
				if(dem[i] > cap[1] + cap[2])
				{
					setup[i][1] = setup[i][1] + 1;
					dem[i] = dem[i] - cap[1];
				}
				else if(dem[i] > cap[1] + cap[3])
				{
					x = rnd(1, 2);
					setup[i][x] = setup[i][x] + 1;
					dem[i] = dem[i] - cap[x];
				}
				else
				{
					x = rnd(1, 3);
					setup[i][x] = setup[i][x] + 1;
					dem[i] = dem[i] - cap[x];
				}
```

```
                        }
                }
                else //se nao aceita carreta
                {
                        while(dem[i] > cap[2])
                        {
                                if(dem[i] > cap[2] + cap[3])
                                {
                                        setup[i][2] = setup[i][2] + 1;
                                         dem[i] = dem[i] - cap[2];
                                }
                                else
                                {
                                        x = rnd(2, 3);
                                        setup[i][x] = setup[i][x] + 1;
                                        dem[i] = dem[i] - cap[x];
                                }
                        }
                }
        }
}


double calcpeso(int r) //calcula o peso de uma rota
{
        int i;
        double peso;

        peso = 0.0;

        for(i = 1; roteiro[r][i] > 0; i++)
                peso = peso + dem[roteiro[r][i]];

        return peso;
}

double calcpesof(int r) //calcula o peso de uma rota na solucao final
{
        int i;
        double peso;

        peso = 0.0;

        for(i = 1; roteiro[r][i] > 0; i++)
                peso = peso + bdem[roteiro[r][i]];

        return peso;
}

double calcciclo(int r) //calcula o ciclo de uma rota
{
        int i;
        double ciclo;

        ciclo = tempop[roteiro[r][0]][roteiro[r][1]];
```

```
	for(i = 2; ((roteiro[r][i-1] > 0) || (roteiro[r][i] > 0)); i++)
	{
		if(hi[roteiro[r][1]] + ciclo + Temp +
(tempop[roteiro[r][i-1]][roteiro[r][i]]) >= hi[roteiro[r][i]])
			ciclo = ciclo + Temp + (tempop[roteiro[r][i-1]][roteiro[r][i]]);
		else
			ciclo = hi[roteiro[r][i]] - hi[0];
	}
	return ciclo;
}

double calcdist(int r) //calcula a distancia de uma rota
{
	int i;
	double distancia;

	distancia = 0.0;

	for(i = 1; (roteiro[r][i-1] > 0 || roteiro[r][i] > 0); i++)
		distancia = distancia + dist[roteiro[r][i-1]][roteiro[r][i]];

	return distancia;
}

int checkcarr (int r)
{
	int i, c;

	c = 1;

	for(i = 1; roteiro[r][i] > 0; i++)
		if(carreta[roteiro[r][i]] == 0)
			c = 0;
	return c;
}

int checkpeso(int x)
{
   int i, p;
	double pesot;

   p = 1;

   pesot = calcpeso(x);

   if(checkcarr(x) == 1)
   {
      if(pesot > cap[1])
		p = 0;
   }
   else
   {
      if(pesot > cap[2])
		p = 0;
```

```
   }

   return p;
}

double calccusto()
{
	int i, j, n;
	double c, d;
	double cust = 0.0;

	/*for(i = 1; i <= N; i++)
		if(roteiro[i][1] > 0)
			for(j = 1; (roteiro[i][j] > 0)||(roteiro[i][j-1]); j++)
				cust = cust + dist[roteiro[i][j-1]][roteiro[i][j]] * (4 - camrota[i]);
	for(i = 1; i <= N; i++)
		for(j = 1; j <= TIPO; j++)
			cust = cust + setup[i][j] * 2 * dist[0][i] * (4 - camrota[i]);*/

	for(i = 1; i <= N; i++)
	{
		c = 0;
		n = 0;
		fretep[i] = 0.0;
		for(j = 1; roteiro[i][j] > 0; j++)
		{
			n++;
			if(custofaixa[roteiro[i][j]][camrota[i]] > c)
			{
				c = custofaixa[roteiro[i][j]][camrota[i]];
				d = delta[roteiro[i][j]][camrota[i]];
			}
		}
		if(n > 0)
		{
			cust = cust + c + (n - 1) * d;
			fretep[i] = fretep[i] + c + (n - 1) * d;
		}
	}

	for(i = 1; i <= N; i++)
		for(j = 1; j <= TIPO; j++)
			cust = cust + setup[i][j] * custofaixa[i][j];
	return cust;
}

void saida()
{
	int i, j, n;

	FILE *arqsaida;
	char nome[100];

	sprintf(nome, "c:/TF/saida.txt");
	arqsaida = fopen(nome,"w");
```

```
for(i = 1; i <= N; i++)
{
	if(broteiro[i][1] > 0)
	{
		for(j = 0; j <= 5; j++)
			fprintf(arqsaida, "%d\n", broteiro[i][j]);
		fprintf(arqsaida, "\n");
	}
}
for(i = 1; i <= N; i++)
	for(j = 1; j <= TIPO; j++)
	{
		n = bsetup[i][j];
		while(n > 0)
		{
			fprintf(arqsaida, "0\n%d\n0\n0\n0\n0\n\n", i);
			n--;
		}
	}


for(i = 1; i <= N; i++)
	fprintf(arqsaida, "0\n0\n0\n0\n0\n0\n\n");

fclose(arqsaida);

// gera arquivo de entrada para o AG
sprintf(nome, "c:/TF/AG.txt");
arqsaida = fopen(nome,"w");

for(i = 1; i <= N; i++)
	for(j = 1; j <= TIPO; j++)
	{
		n = bsetup[i][j];
		while(n > 0)
		{
			fprintf(arqsaida, "%d\t", j);
			fprintf(arqsaida, "%d\t0\t0\t0\n", i);
			fprintf(arqsaida, "0\t");
			fprintf(arqsaida, "%lf\t0\t0\t0\n", cap[j]);
			n--;
		}
	}

for(i = 1; i <= N; i++)
	if(broteiro[i][1] > 0)
	{
		fprintf(arqsaida, "%d\t", bcamrota[i]);
		for(j = 1; j <= TAM; j++)
			fprintf(arqsaida, "%d\t", broteiro[i][j]);
		fprintf(arqsaida, "\n");
		fprintf(arqsaida, "0\t");
		for(j = 1; j <= TAM; j++)
			fprintf(arqsaida, "%lf\t", bdem[broteiro[i][j]]);
```

```
                fprintf(arqsaida, "\n");
        }

for(i = 1; i <= 2*N; i++)
        fprintf(arqsaida, "0\t0\t0\t0\n");

fclose(arqsaida);

// gera arquivo com os caminhoes utilizados
sprintf(nome, "c:/TF/cam.txt");
arqsaida = fopen(nome,"w");

for(i = 1; i <= N; i++)
        if(broteiro[i][1] > 0)
                fprintf(arqsaida, "%d\n\n\n\n\n\n\n\n", bcamrota[i]);

for(i = 1; i <= N; i++)
        for(j = 1; j <= TIPO; j++)
        {
                n = bsetup[i][j];
                while(n > 0)
                {
                        fprintf(arqsaida, "%d\n\n\n\n\n\n\n\n", j);
                        n--;
                }
        }

for(i = 1; i <= N; i++)
        fprintf(arqsaida, "0\n\n\n\n\n\n\n\n");
fclose(arqsaida);

sprintf(nome, "c:/TF/rotas.txt");
arqsaida = fopen(nome,"w");

for(i = 1; i <= N; i++)
        for(j = 1; j <= TIPO; j++)
        {
                n = bsetup[i][j];
                while(n > 0)
                {
                        fprintf(arqsaida, "%d\t", j);
                        fprintf(arqsaida, "%d\n", i);
                        n--;
                }
        }

for(i = 1; i <= N; i++)
        if(broteiro[i][1] > 0)
        {
                fprintf(arqsaida, "%d\t", bcamrota[i]);
                for(j = 1; j <= TAM; j++)
                        if(broteiro[i][j] > 0)
                                fprintf(arqsaida, "%d\t", broteiro[i][j]);
                fprintf(arqsaida, "\n");
        }
```

```
	fclose(arqsaida);
}

int main()
{
	int i, j, k, r, g, num, n, nn;
	int tipo, tipo2, rota, rota2, posicao, posicao2;
	int norotas[NMAX+1][3];
	int checkh, checkc;

	double peso[NMAX+1];
	double pesot;
	double ciclo[NMAX+1];
	double economia[NMAX+1][NMAX+1];

	double ciclot;
	double custo;
	double bestc;
	double cf;

	FILE *arqentrada;
	char nome[100];

	 for(i = 0;i <= NMAX; i++)
		d[i] = 0.0;

	sprintf(nome, "c:/TF/Dados22-02.txt"); // dados das lojas
	arqentrada = fopen(nome,"r");
	fscanf(arqentrada, "%d", &N);
	for(i=0; i<= N; i++)
	{
		fscanf(arqentrada, "%lf", &X[i]);
		fscanf(arqentrada, "%lf", &Y[i]);
		fscanf(arqentrada, "%lf", &d[i]);
		fscanf(arqentrada, "%lf", &custofaixa[i][1]);
		fscanf(arqentrada, "%lf", &custofaixa[i][2]);
		fscanf(arqentrada, "%lf", &custofaixa[i][3]);
		fscanf(arqentrada, "%lf", &delta[i][1]);
		fscanf(arqentrada, "%lf", &delta[i][2]);
		fscanf(arqentrada, "%lf", &delta[i][3]);
		fscanf(arqentrada, "%lf", &hi[i]);
		fscanf(arqentrada, "%lf", &hf[i]);
		fscanf(arqentrada, "%d", &carreta[i]);
	}
	fclose (arqentrada);

	printf("Numero ");
   scanf("%d", &num);
   srand(num);

   bestc = 999999999.9;

   cap[1] = 28.0;
	cap[2] = 14.0;
	cap[3] = 7.0;
```

```
for(i = 0;i <= N; i++)
		for(j = 0; j <= N; j++)
		{
			dist[i][j] = 0.0;
			economia[i][j] = 0.0;
		}

for(i = 0;i <= (N*(N-1))/2; i++)
	{
		listae[i]= 0.0;
		listai[i] = 0;
		listaj[i] = 0;
	}

	for(i = 1; i <= N; i++)
	{
		bdem[i] = 0.0;
		fretep[i] = 0.0;
	}

printf("Calculando Distancias...\n");
calcdistxy();
	calctempop();

printf("Calculando Economias...\n");
/*for(i = 1;i <= N; i++) // calcula economias (distancia)
		for(j = 1; j <= N; j++)
			if(i != j)
				economia[i][j] = dist[0][i] + dist[j][0] - dist[i][j];*/

	for(i = 1;i <= N; i++) // calcula economias 2 (custo de frete)
		for(j = 1; j <= N; j++)
			if(i != j)
			{
				if(custofaixa[i][1] >= custofaixa[j][1])
				{
					economia[i][j] = custofaixa[j][1] - delta[i][1];
				}
				else
				{
					economia[i][j] = custofaixa[i][1] - delta[j][1];
				}
			}

printf("Gerando Lista...\n");
for(k = 1; k <= (N*(N-1)/2); k++)
	{
		for(i = 1; i <= N; i++)
			for(j = i+1; j <=N; j++)
				if(economia[i][j] > listae[k])
				{
					listai[k] = i;
					listaj[k] = j;
					listae[k] = economia[i][j];
```

```
                                        }
                        economia[listai[k]][listaj[k]] = 0.0;
        }

        for(g = 1; g <= GER; g++)
        {

        for(i = 0; i <= N; i++)
        {
                        peso[i] = 0.0;
                        ciclo[i] = 0.0;
                        horario[i] = 0.0;
        }

  tipo = 0;
  tipo2 = 0;
  rota = 0;
  rota2 = 0;
  posicao = 0;
  posicao2 = 0;
  checkc = 0;
  checkh = 0;

  for(i = 0; i <= N; i++)
                        for(j = 0; j <= N+1; j++)
                                roteiro[i][j] = 0;

for(i = 1; i <= N; i++)
       dem[i] = d[i];

for(i = 1; i <= N; i++)
      for(j = 1; j <= TIPO; j++)
           setup[i][j] = 0;

 if(g == 1)
        entregasini();
 else
        entregasini();

        for(i = 0; i <= N; i++)
                        for( j = 0; j <= 3; j++)
                                norotas[i][j] = 0;

        r = 1;
        for(k = 1; k <= (N*(N-1)/2); k++)
        {
          if((randomico(1) < PH) || (g == 1))
          {

                        if((norotas[listai[k]][0] >= 2) && (norotas[listaj[k]][0] >= 2))
                        {
                                tipo = norotas[listai[k]][0];
                                rota = norotas[listai[k]][1];
                                posicao = norotas[listai[k]][2];
                                tipo2 = norotas[listaj[k]][0];
```

```
rota2 = norotas[listaj[k]][1];
posicao2 = norotas[listaj[k]][2];

if(tipo == 3) //se o primeiro é final de rota
{
	for(i = 1; roteiro[rota][i] > 0; i++)
		roteiro[0][i] = roteiro[rota][i];
	if(tipo2 == 2)
		for(i = 1; roteiro[rota2][i] > 0; i++)
			roteiro[0][posicao + i] = roteiro[rota2][i];
	if(tipo2 == 3)
	{
		j = posicao + 1;
		for(i = posicao2; i >= 1; i--)
		{
			roteiro[0][j] = roteiro[rota2][i];
			j++;
		}

	}
}
if(tipo == 2) //se o primeiro é início de rota
{
	if(tipo2 == 2)
	{
		j = 1;
		for(i = contno(rota2); i >= 1; i--)
		{
			roteiro[0][j] = roteiro[rota2][i];
			j++;
		}
		for(i = 1; roteiro[rota][i] > 0; i++)
			roteiro[0][j+i-1] = roteiro[rota][i];
	}
	if(tipo2 == 3)
	{
		for(i = 1; roteiro[rota2][i] > 0; i++)
			roteiro[0][i] = roteiro[rota2][i];
		for(i = 1; roteiro[rota][i] > 0; i++)
			roteiro[0][posicao2 + i] = roteiro[rota][i];
	}
}
pesot = calcpeso(0);
ciclot = calcciclo(0);
calchorario(0);
checkh = 1;
for(i = 1; roteiro[0][i] > 0; i++)
	if(horario[roteiro[0][i]] > hf[roteiro[0][i]])
		checkh = 0;

checkc = checkpeso(0);

if((checkc == 1)&&(ciclot <= T)&&(checkh == 1))
{
```

```
                        for(i = 1; roteiro[0][i] > 0; i++)
                        {
                                roteiro[rota][i] = roteiro[0][i];
                                norotas[roteiro[rota][i]][2] = i;
                                norotas[roteiro[rota][i]][1] = rota;
                                norotas[roteiro[rota][i]][0] = 1;
                                if(roteiro[0][i+1] == 0)
                                        norotas[roteiro[rota][i]][0] = 3;
                        }
                        norotas[roteiro[rota][1]][0] = 2;

                        for(i = 1; roteiro[rota2][i] > 0; i++)
                                roteiro[rota2][i] = 0;

                        //peso[rota] = pesot;
                        ciclo[rota]= ciclot;
                }
                for(i = 1;i <= N+1; i++) //apaga roteiro temporário
                        roteiro[0][i] = 0;
        }
        else if(norotas[listai[k]][0] >= 2 && norotas[listaj[k]][0] == 0)
        {
                tipo = norotas[listai[k]][0];
                rota = norotas[listai[k]][1];
                posicao = norotas[listai[k]][2];

                if(tipo == 3)
                {
                        roteiro[rota][posicao+1] = listaj[k];
                        pesot = calcpeso(rota);
                        ciclot = calcciclo(rota);
                        calchorario(rota);
                        checkh = 1;
                        for(i = 1; roteiro[rota][i] > 0; i++)
                                if(horario[roteiro[rota][i]] > hf[roteiro[rota][i]])
                                        checkh = 0;

                        checkc = checkpeso(rota);
                        if((checkc == 1)&&(ciclot <= T)&&(checkh == 1))
                        {
                                norotas[listaj[k]][0] = 3;
                                norotas[listaj[k]][1] = rota;
                                norotas[listaj[k]][2] = posicao + 1;
                                norotas[listai[k]][0] = 1;
                                //peso[rota] = pesot;
                                ciclo[rota]= ciclot;
                        }
                        else
                                roteiro[rota][posicao + 1] = 0;
                }
                if(tipo == 2)
                {
                        for(i = N; i >= 2; i--)
                                roteiro[rota][i] = roteiro[rota][i-1];
                        roteiro[rota][1] = listaj[k];
```

```
                    pesot = calcpeso(rota);
                    ciclot = calcciclo(rota);
                    calchorario(rota);
                    checkh = 1;
                    for(i = 1; roteiro[rota][i] > 0; i++)
                            if(horario[roteiro[rota][i]] > hf[roteiro[rota][i]])
                                    checkh = 0;

                    checkc = checkpeso(rota);

                    if((checkc == 1)&&(ciclot <= T)&&(checkh == 1))
                    {
                            norotas[listaj[k]][0] = 2;
                            norotas[listaj[k]][1] = rota;
                            norotas[listaj[k]][2] = 1;
                            norotas[listai[k]][0] = 1;
                            for(i = 2; roteiro[rota][i] > 0; i++)
                                    norotas[roteiro[rota][i]][2]++;
                            ciclo[rota] = ciclot;
                    }
                    else
                            for(i = 1; roteiro[rota][i] > 0; i++)
                                    roteiro[rota][i] = roteiro[rota][i+1];
            }
    }
    else if(norotas[listaj[k]][0] >= 2 && norotas[listai[k]][0] == 0)
    {
            tipo = norotas[listaj[k]][0];
            rota = norotas[listaj[k]][1];
            posicao = norotas[listaj[k]][2];

            if(tipo == 3)
            {
                    roteiro[rota][posicao + 1] = listai[k];
                    pesot = calcpeso(rota);
                    ciclot = calcciclo(rota);
                    calchorario(rota);
                    checkh = 1;
                    for(i = 1; roteiro[rota][i] > 0; i++)
                            if(horario[roteiro[rota][i]] > hf[roteiro[rota][i]])
                                    checkh = 0;

                    checkc = checkpeso(rota);



                    if((checkc == 1)&&(ciclot <= T)&&(checkh == 1))
                    {

                            norotas[listai[k]][0] = 3;
                            norotas[listai[k]][1] = rota;
                            norotas[listai[k]][2] = posicao + 1;
                            norotas[listaj[k]][0] = 1;
                            //peso[rota] = pesot;
                            ciclo[rota]= ciclot;
```

```
                        }
                        else
                                roteiro[rota][posicao + 1] = 0;
                }
                if(tipo == 2)
                {
                        for(i = N; i >= 2; i--)
                                roteiro[rota][i] = roteiro[rota][i-1];
                        roteiro[rota][1] = listai[k];
                        pesot = calcpeso(rota);
                        ciclot = calcciclo(rota);
                        calchorario(rota);
                        checkh = 1;
                        for(i = 1; roteiro[rota][i] > 0; i++)
                                if(horario[roteiro[rota][i]] > hf[roteiro[rota][i]])
                                        checkh = 0;

                        checkc = checkpeso(rota);

                        if((checkc == 1)&&(ciclot <= T)&&(checkh == 1))
                        {
                                norotas[listai[k]][0] = 2;
                                norotas[listai[k]][1] = rota;
                                norotas[listai[k]][2] = 1;
                                norotas[listaj[k]][0] = 1;
                                for(i = 2; roteiro[rota][i] > 0; i++)
                                        norotas[roteiro[rota][i]][2]++;
                                //peso[rota] = pesot;
                                ciclo[rota] = ciclot;
                        }
                        else
                                for(i = 1; roteiro[rota][i] > 0; i++)
                                        roteiro[rota][i] = roteiro[rota][i+1];
                }
        }
        else if((norotas[listai[k]][0] == 0)&&(norotas[listaj[k]][0] == 0))
        {
                roteiro[0][1] = listai[k];
                roteiro[0][2] = listaj[k];
                pesot = calcpeso(0);
                ciclot = calcciclo(0);
                calchorario(0);

                checkh = 1;
                for(i = 1; roteiro[0][i] > 0; i++)
                        if(horario[roteiro[0][i]] > hf[roteiro[0][i]])
                                checkh = 0;

checkc = checkpeso(0);

                if((checkc == 1)&&(ciclot <= T)&&(checkh == 1))
                {
                        norotas[listai[k]][0] = 2;
                        norotas[listaj[k]][0] = 3;
                        norotas[listai[k]][1] = r;
```

```
                                        norotas[listaj[k]][1] = r;
                                        norotas[listai[k]][2] = 1;
                                        norotas[listaj[k]][2] = 2;
                                        roteiro[r][1] = roteiro[0][1];
                                        roteiro[r][2] = roteiro[0][2];
                                        ciclo[r] = ciclot;
                                        r++;
                                }
                                roteiro[0][1] = 0;
                                roteiro[0][2] = 0;
                        }
                }
        }
for(i = 1; i <= N-1; i++) // elimina roteiros vazios
   if(roteiro[i][1] == 0)
   {
      for(j = i+1; ((roteiro[j][1] == 0)&&(j < N)); j++)
           j = j;
      for(k = 1; roteiro[j][k] > 0; k++)
      {
           roteiro[i][k] = roteiro[j][k];
           roteiro[j][k] = 0;
           peso[i] = peso[j];
           ciclo[i] = ciclo[j];
      }
   }

for(i = 1; roteiro[i][1] > 0; i++)
     peso[i] = calcpeso(i);

r = 0;
for(i = 1; i <= N; i++)
   if(roteiro[i][1] > 0) // conta o numero de rotas
      r++;

      for(i = 1; i <= N; i++) //cria as rotas individuais
                if(norotas[i][0] == 0)
                {
                        roteiro[r+1][1] = i;
                        peso[r+1] = calcpeso(r+1);
                        ciclo[r+1] = calcciclo(r+1);
                        r++;
                }

      for(i = 1; i <= N; i++) //atribui os caminhoes adequados a cada rota
      {
                if(roteiro[i][1] > 0)
                {
                        camrota[i] = 1;
                        if(peso[i] <= cap[2])
                                camrota[i] = 2;
                        if(peso[i] <= cap[3])
                                camrota[i] = 3;
                }
      }
```

```
		custo = calccusto();

		if(custo < bestc)
		{
			bestc = custo;
			for(i = 1; i <= N; i++)
			{
				for(j = 1; j <= TIPO; j++)
					bsetup[i][j] = setup[i][j];
				bcamrota[i] = camrota[i];
				for(j = 0; j <= N; j++)
					broteiro[i][j] = roteiro[i][j];
				bdem[i] = dem[i];
				bfretep[i] = fretep[i];
			}
		}

		printf("G = %d\tTotal: %lf\n", g, bestc);

		}
		saida();

		for(i = 1; i <= N; i++) // imprime os roteiros formados
			{
				if(broteiro[i][1] > 0)
				{
					printf("R%d - 0 ", i);
					for(j = 1; (broteiro[i][j] > 0 || broteiro[i][j-1] > 0) ; j++)
						printf("%d ", broteiro[i][j]);
					printf("\n");
				}
			}
		cf = 0;
		for(i = 1; i <= N; i++)
		for(j = 1; j <= TIPO; j++)
			{
				n = bsetup[i][j];
				while(n > 0)
				{
					printf("%d, F = %lf\n", i, custofaixa[i][j]);
					n--;
				}
			}

   for(i = 1; i <= N; i++)
        for(j = 1; j <= N+1; j++)
            roteiro[i][j] = broteiro[i][j];

		for(i = 1; broteiro[i][1] > 0; i++)
			printf("R%d - T = %d, P = %lf, T = %lf, D = %lf, F = %lf\n", i, bcamrota[i],
calcpesof(i), calcciclo(i), calcdist(i), bfretep[i]);
		printf("Total: %lf\n", bestc);
		return 0;
}
```