
Universidade Estadual de Campinas - UNICAMP

Instituto de Matemática, Estatística e Computação Científica

Departamento de Matemática Aplicada

Dissertação de Mestrado:

Metaheurística para a Solução de Problemas de
Roteamento de Veículos com Janela de Tempo

Autora: Heloisa Passarelli Vieira


Orientador: Francisco A. Magalhães Gomes

2 de maio de 2013

**METAHEURÍSTICA PARA A SOLUÇÃO
DE PROBLEMAS DE ROTEAMENTO DE VEÍCULOS
COM JANELA DE TEMPO**

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por **Heloísa Passarelli Vieira** e aprovada pela comissão julgadora.

Campinas, 11 de dezembro de 2008.



Prof. Dr. Francisco A. M. Gomes Neto
Orientador.

Banca Examinadora:

Prof. Dr. Francisco de Assis Magalhães Gomes Neto (IMECC – UNICAMP)

Prof^a. Dr^a. Franklina Maria Bragion de Toledo (ICMC – USP)

Prof. Dr. Aurélio Ribeiro Leite de Oliveira (IMECC – UNICAMP)

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para obtenção do Título de MESTRE em Matemática Aplicada.

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP**

Bibliotecária: Crislene Queiroz Custódio – CRB8 / 7966

Vieira, Heloisa Passarelli

V673m Metaheurística para a solução de problemas de roteamento de
veículos com janela de tempo / Heloisa Passarelli Vieira – Campinas, [S.P. :
s.n.], 2008.

Orientador : Francisco de Assis Magalhães Gomes Neto
Dissertação (Mestrado) - Universidade Estadual de Campinas, Instituto
de Matemática, Estatística e Computação Científica.

1. Problema de roteamento de veículos. 2. Janela de tempo. 3.
Algoritmos genéticos. 4. Metaheurística. I. Gomes Neto, Francisco de Assis
Magalhães. II. Universidade Estadual de Campinas. Instituto de
Matemática, Estatística e Computação Científica. III. Título.

(cqc/imecc)

Título em inglês: Metaheuristics for the solution of vehicle routing problems with time windows

Palavras-chave em inglês (Keywords): 1. Vehicle routing problem. 2. Time Window. 3. Genetic algorithms. 4. Metaheuristic.

Área de concentração: Pesquisa Operacional

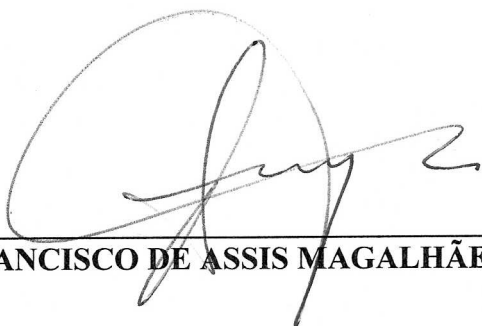
Titulação: Mestre em Matemática Aplicada

Banca examinadora: Prof. Dr. Francisco de Assis Magalhães Gomes Neto (IMECC-Unicamp)
Prof. Dr. Aurélio Ribeiro Leite de Oliveira (IMECC-Unicamp)
Prof. Dra. Franklina Maria Bragion de Toledo (ICMC- USP)

Data da defesa: 11/12/2008

Programa de Pós-Graduação: Mestrado em Matemática Aplicada

**Dissertação de Mestrado defendida em 11 de dezembro de 2008 e aprovada
pela Banca Examinadora composta pelos Profs. Drs.**



Prof.(a). Dr(a). FRANCISCO DE ASSIS MAGALHÃES GOMES NETO



Prof. (a). Dr (a). FRANKLINA MARIA BRAGION DE TOLEDO



Prof. (a). Dr (a). AURELIO RIBEIRO LEITE DE OLIVEIRA

RESUMO

Nos últimos anos, diversas heurísticas e meta-heurísticas foram propostas para o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT), cujo objetivo é determinar a rota a ser seguida por uma frota de veículos para servir um número de clientes em um dado intervalo de tempo, sem violar a capacidade dos veículos. Cada cliente é visitado por exatamente um veículo e somente uma vez. Esta dissertação apresenta um estudo das técnicas utilizadas para o PRVJT, dando ênfase para os Algoritmos Genéticos. Diversos tipos de cruzamento e esquemas de mutação, além de outras técnicas avançadas, tal como o *Hill-Climbing*, são analisados. Para o algoritmo que implementamos, são apresentados vários resultados numéricos baseados em um conjunto de 56 problemas, cada qual com 100 clientes, proposto por Solomon. O desempenho do algoritmo que implementamos também é comparado aos melhores resultados publicados na literatura.

Palavras Chave: Roteamento de veículos com janela de tempo, Metaheurísticas, Algoritmos Genéticos.

ABSTRACT

In recent years, several heuristic and metaheuristic methods were proposed for the Vehicle Routing Problem with Time Windows (VRPTW). The objective of the problem is to serve a set of customers within a given time interval, without violating the capacity of the vehicles. Each customer must be visited once and by only one vehicle. This dissertation presents a survey on the techniques used to solve the VRPTW, with emphasis on the genetic algorithms. Several crossover and mutation schemes, as well as other advanced techniques, such as the *Hill-Climbing* are analyzed. Numerical results based on Solomon's 56 VRPTW 100-customer instances are presented for the algorithm implemented here. The performance of our algorithm is also compared with the best results published in the specialized literature.

Keywords: Vehicle Routing Problem with Time Windows, Metaheuristics, Genetic Algorithms.

Sumário

1	Introdução	1
2	O problema de roteamento de veículos com janela de tempo	5
2.1	O Problema Capacitado.	6
2.2	Extensões do Problema de Roteamento de Veículos.	8
2.3	Problema de Roteamento de Veículos com Janela de Tempo.	12
3	Técnicas de Solução	15
3.1	Métodos Exatos.	15
3.1.1	<i>Branch-and-Bound</i>	16
3.1.2	<i>Branch-and-Cut</i>	16
3.1.3	<i>Branch-and-Price</i>	17
3.2	Heurísticas.	17
3.2.1	Heurísticas de construção.	18
3.2.2	Heurísticas de Refinamento.	22
3.3	Metaheurísticas.	31
4	Algoritmos Genéticos	39
4.1	Algoritmos Evolucionários.	40
4.2	Características Gerais dos Algoritmos Genéticos.	40
4.2.1	Operadores Genéticos.	41
4.2.2	Parâmetros Genéticos.	42
4.2.3	Estrutura do Algoritmo Genético.	43

4.3	Teoria de Convergência dos Algoritmos Genéticos.	47
5	Algoritmo Genético para Roteamento com Janela de Tempo	49
5.1	Representação do cromossomo.	49
5.2	População Inicial.	51
5.2.1	Construção da Solução Inicial.	52
5.2.2	Geração da População.	55
5.3	Seleção.	57
5.4	Reprodução.	58
5.4.1	Cruzamento.	59
5.4.2	Mutação.	65
5.5	Procedimentos de Pós-Otimização.	66
5.5.1	<i>Hill Climbing</i>	66
5.5.2	Recuperação (<i>Recovery</i>).	67
5.6	O Algoritmo.	68
6	Experimentos Computacionais	69
6.1	Problemas utilizados nos testes.	69
6.2	Perfis de desempenho.	70
6.3	Testes executados no Matlab.	71
6.3.1	Tipo de cruzamento.	72
6.3.2	Vizinhança da solução inicial oriunda do método PFIH.	73
6.3.3	Cliente inicial das rotas geradas pelo algoritmo PFIH.	75
6.3.4	<i>Hill climbing</i>	76
6.4	Testes com a implementação em c++.	80
6.4.1	Comparação com o algoritmo desenvolvido no Matlab.	81
6.4.2	Ponderação entre distância percorrida e número de veículos.	83
6.4.3	Comparação com outros algoritmos.	87
7	Considerações Finais	93

Introdução

O grande crescimento populacional, a descentralização dos pontos de venda e o aumento da variedade de produtos têm provocado o crescimento e o aumento da complexidade da rede de distribuição de bens e serviços. Ao mesmo tempo, afim de evitar o caos urbano, principalmente nos grandes centros, as companhias de engenharia de tráfego impõem uma série de restrições tanto ao tamanho dos veículos quanto ao horário das operações de entrega e coleta de produtos. Além disso, a globalização e o aumento da informatização têm tornado os clientes mais exigentes, tanto no que concerne à qualidade, quanto aos prazos de entrega do produto. Sendo assim, para as empresas, a distribuição e os processos logísticos se tornaram importantes nas operações gerenciais, pois para se manter competitivo e conquistar um mercado cada vez mais concorrido, tornou-se fundamental a adoção de processos mais rápidos e adaptados ao perfil do cliente. É nesse contexto que a logística vem-se consolidando como um dos elementos chaves na estratégia competitiva das empresas, que passaram a atribuir mais valor ao cliente, pois a satisfação destes passou a ser o elemento fundamental no mercado atual.

Dentre todos os processos envolvidos na cadeia logística, o transporte é aquele que absorve a maior parcela do custo, atingindo de um a dois terços do valor total [24]. Reduzir esse custo significa diminuir o preço final do produto acabado, além de aumentar a lucratividade. A conquista de mercados cada vez mais concorridos desencadeou um aumento do interesse em pacotes de otimização, acarretando um aumento do número de pesquisas e artigos publicados,

em especial no estudo de roteamento de veículos. Essas pesquisas propiciaram um melhor gerenciamento do processo de distribuição, levando à redução de custos tanto pelo emprego de rotas mais curtas, como pela redução das penalidades devidas ao atraso da entrega.

O Problema de Roteamento de Veículos (PRV) é, de uma forma bem geral, um problema que envolve a distribuição de bens (ou serviços) de um depósito central para usuários finais (clientes). Essa distribuição é feita por uma frota de veículos que partem do depósito, passam pelos clientes, satisfazendo integralmente suas demandas, e retornam ao depósito. O objetivo do problema é a determinação das rotas com custo mínimo para os veículos, sem violar os limites de capacidade destes.

O PRV foi introduzido por Dantzig e Ramser em 1959 [33]. Em 1964, Clarke e Wright [29] propuseram um algoritmo guloso que forneceu resultados melhores que aqueles obtidos por Dantzig e Ramser. Desde então, o problema vem sendo muito estudado, principalmente por sua alta complexidade e pela grande variedade de problemas reais a ele associados. Como exemplo de aplicações podemos citar:

- a entrega postal;
- a entrega, em domicílio, de produtos comprados nas lojas de varejo ou pela internet;
- a distribuição de produtos dos centros de distribuição (CD) de atacadistas para lojas do varejo;
- a escolha de rotas para ônibus escolares ou de empresas.

O problema de roteamento é um problema de programação inteira que pertence à categoria dos problemas NP-difíceis. Sendo assim, dependendo do tamanho do problema, não existe um algoritmo de otimização capaz de obter a solução ótima exata do problema em tempo hábil. Dessa forma, se faz necessário o uso de métodos heurísticos e meta-heurísticos que sejam capazes de chegar perto da solução ótima, consumindo tempo e esforço computacionais relativamente pequenos quando comparados aos gastos dos métodos exatos.

Esse trabalho é organizado da seguinte forma: No Capítulo 2, descrevemos o problema de roteamento de veículos e suas variações, destacando o problema com janela de tempo. No Capítulo

3, apresentamos as principais técnicas de solução utilizadas para o problema de roteamento com janela de tempo. No Capítulo 4, damos uma atenção especial aos Algoritmos genéticos para, no Capítulo 5, introduzirmos o algoritmo que utilizamos para resolver o problema. Finalmente, no Capítulo 6, mostramos alguns resultados computacionais.

O problema de roteamento de veículos com janela de tempo

Suponhamos que uma frota de veículos esteja disponível para o transporte de mercadorias demandadas ou ofertadas por um conjunto de clientes. Suponhamos, também, que cada veículo esteja situado em um depósito. O Problema de Roteamento de Veículos (PRV) consiste em determinar a rota a ser atribuída a cada veículo, de modo que a demanda ou oferta de todos os clientes seja satisfeita, e que cada veículo regresse ao depósito de origem ao final do período considerado. O objetivo é minimizar o custo total, definido pela soma dos custos dos roteiros.

O Problema de Roteamento de Veículos Capacitado (PRVC) é a versão mais simples do problema. Nela, todos os clientes possuem demandas determinísticas, ou seja, conhecidas previamente, que devem ser atendidas integralmente por apenas um veículo. Todos os veículos são semelhantes e partem de um único depósito. Somente uma restrição de capacidade é imposta ao problema. Essa restrição estabelece que a soma das demandas de todos os clientes pertencente a uma rota não deve superar a capacidade do veículo a ela designado. O PRVC deu origem a diversos outros problemas, motivo pelo qual iremos estudá-lo em primeiro lugar, apresentando, em seguida, as suas variações.

2.1 O Problema Capacitado.

Seja um grafo $G = (V, A)$ completo, em que A é um conjunto de arcos, que representam os caminhos que ligam os clientes entre si e estes ao depósito, e $V = 0, \dots, n$ denota um conjunto de $n+1$ vértices. Convencionamos que o vértice 0 representa o depósito e os outros simbolizam os clientes. A cada arco (i, j) é associado um custo não negativo, $c_{i,j}$, que representa o custo de viagem do vértice i ao vértice j . Na maioria dos casos, os custos dos arcos satisfazem a desigualdade triangular,

$$c_{i,k} + c_{k,j} \geq c_{i,j},$$

para $i, k, j \in V$. Quando o custo do arco (i, j) é igual ao custo do arco (j, i) , dizemos que o problema é *simétrico*. Caso contrário, ele é dito *assimétrico*.

Um conjunto K de veículos idênticos, com capacidade C , é alocado a um único depósito. A cada cliente i é associado uma demanda não negativa, m_i . Para o depósito, definimos $m_0 = 0$.

O PRVC consiste em encontrar um conjunto de exatamente K rotas, cada uma percorrida por um veículo, de modo a minimizar o custo total de transporte e a satisfazer as seguintes restrições:

- cada rota deve ter início e fim no depósito;
- cada cliente deve ser visitado apenas uma vez e somente por um veículo;
- a soma das demandas dos clientes incluídos em uma rota não deve exceder a capacidade do veículo.

Para definir o problema, utilizamos Kn^2 variáveis binárias, dadas por

$$x_{ijk} = \begin{cases} 1, & \text{se o veículo } k \text{ trafega no trecho } (i, j); \\ 0, & \text{caso contrário.} \end{cases}$$

para $i, j \in \{0, 1, \dots, n\}$, $i \neq j$ e $k \in \{1, \dots, K\}$. A formulação matemática do PRVC é apresentada abaixo.

$$\text{Minimizar} \quad \sum_{i=0}^n \sum_{\substack{j=0 \\ j \neq i}}^n \sum_{k=1}^K c_{ij} x_{ijk} \quad (2.1)$$

$$\text{sujeito a} \quad \sum_{k=1}^K \sum_{j=1}^n x_{0jk} = K \quad (2.2)$$

$$\sum_{j=1}^n x_{0jk} = \sum_{j=1}^n x_{j0k} = 1, \quad k = 1, \dots, K \quad (2.3)$$

$$\sum_{k=1}^K \sum_{j=0}^n x_{ijk} = 1, \quad i = 1, \dots, n \quad (2.4)$$

$$\sum_{j=0}^n x_{ijk} - \sum_{j=0}^n x_{jik} = 0, \quad k = 1, \dots, K, \quad i = 1, \dots, n \quad (2.5)$$

$$\sum_{k=1}^K \sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - v(S), \quad \forall S \subseteq V \setminus \{0\}, \quad |S| \geq 2 \quad (2.6)$$

$$\sum_{i=1}^n m_i \sum_{\substack{j=0 \\ j \neq i}}^n x_{ijk} \leq C, \quad k = 1, \dots, K, \quad (2.7)$$

$$x_{ijk} \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, n, \quad k = 1, \dots, K. \quad (2.8)$$

A equação (2.2) garante que exatamente K veículos sairão do depósito, enquanto (2.3) assegura que cada rota tenha início e fim no depósito. As restrições (2.4) e (2.5) garantem que cada cliente seja visitado exatamente uma vez, e que o veículo que chega em um cliente saia dele. A restrição (2.6) evita que seja formado um subciclo que não inclua o depósito. Nesta restrição, $v(S)$ representa o número mínimo de veículos necessário para atender o conjunto de clientes S .

Por exigir que o número de veículos usados para atender os clientes do conjunto S não seja inferior a $v(S)$, o mínimo necessário, a família de restrições (2.6) assegura, indiretamente, que a capacidade dos veículos não seja violada. Entretanto, para dar maior clareza à formulação do problema e facilitar as referências futuras, preferimos manter explícita a restrição de capacidade dos veículos, motivo pelo qual introduzimos (2.7).

O problema capacitado de roteamento possui várias outras formulações, muitas das quais incluindo apenas n^2 variáveis. De fato, pode-se notar que, como todos os veículos possuem a

mesma capacidade e como as rotas só se encontram no depósito, não é necessário indicar qual veículo percorrerá cada rota. Desta forma, poderíamos utilizar x_{ij} simplesmente, em lugar de x_{ijk} .

Cabe notar que a adoção da formulação (2.1)-(2.8), que tem mais variáveis e restrições do que o estritamente necessário, não provocará uma perda de eficiência de nosso algoritmo, uma vez que não trabalharemos com um método direto, mas com um algoritmo genético. Como veremos, a definição adequada dos cromossomos é suficiente para garantir que todas as restrições acima sejam satisfeitas.

2.2 Extensões do Problema de Roteamento de Veículos.

Pode-se perceber que o PRVC, por ser muito simples, não é capaz de representar todas as situações cotidianas enfrentadas pelos setores de logística das empresas de distribuição de mercadorias e serviços. Assim, muitas vezes é necessário introduzir neste problema algumas restrições associadas aos clientes, aos veículos ou aos depósitos, para que ele reflita as características dos problemas reais.

Como exemplo, na prática, os clientes podem requerer que:

- diferentes tipos de produtos sejam entregues;
- apenas um determinado subconjunto dos veículos seja usado para servi-los, em função, por exemplo, de limitações de acesso, ou do tipo de mercadoria transportada;
- o serviço de carregamento ou descarregamento das mercadorias seja executado em um tempo predefinido;
- a entrega dos produtos seja feita em um período determinado do dia ou mês (janela de tempo), de maneira a respeitar os horários de funcionamento de seus estabelecimentos, as limitações de tráfego e os prazos de entrega exigidos pelos consumidores por eles atendidos;
- produtos em excesso sejam devolvidos, quer por terem perdido a validade, como ocorre com jornais e revistas, quer por não terem sido bem recebidos pelos consumidores.

Os veículos, por sua vez,

- podem ser de diferentes tipos, ou possuírem capacidades diferentes;
- podem ser subdivididos em compartimentos, permitindo o transporte de diferentes produtos em diversas quantidades;
- podem suportar um tempo máximo de operação, antes de serem submetidos a revisão.

Além disso, é possível que exista mais de um depósito, de modo que:

- a rota de cada veículo possa ter início e fim em um depósito específico, diferente do utilizado por outros veículos;
- cada veículo possa partir de um depósito e terminar sua rota em outro depósito.

Outras variantes do PRVC podem ser obtidas se considerarmos, por exemplo, que:

- o número de veículos utilizados não precisa ser previamente determinado (ou seja, há um grande número de veículos à disposição);
- a demanda é estocástica.

Naturalmente, cada uma dessas modificações eleva a complexidade do modelo, pois acarreta em novas restrições. Assim, geralmente não é possível levar todas em consideração concomitantemente. Descrevemos abaixo os principais problemas surgidos a partir da combinação de algumas dessas variações.

1. **Problema de Roteamento de Veículos com Prioridade para a Entrega (PRVPE) - Vehicle Routing Problem with Backhauls (VRPB).**

O Problema de Roteamento com Prioridade para a Entrega é uma extensão do problema capacitado na qual os clientes são divididos em dois subconjuntos. O primeiro contém os clientes aos quais é preciso entregar produtos. Em inglês, esses clientes são denominados *linehaul customers*. O segundo subconjunto é formado pelos clientes que desejam enviar

produtos ao depósito. Estes são denominados *backhaul costumers*. Assim, a cada cliente é associada uma demanda não negativa, que pode ser de entrega ou de coleta. Os clientes que receberão mercadorias devem ser atendido antes dos clientes que terão seus produtos coletados, caracterizando uma relação de precedência.

Além de incluir uma restrição que define essa precedência, o PRVPE também possui uma restrição de capacidade levemente diferente daquela usada no PRVC, já que tanto a carga a ser entregue quanto a coletada não deve ultrapassar a capacidade do veículo.

2. Problema de Roteamento de Veículos com Coleta e Entrega (PRVCE) - Vehicle Routing Problem with Pickup and Delivery (VRPPD)

Assim como o PRVPE, o Problema de Roteamento de Veículos com Coleta e Entrega permite que parte dos clientes receba mercadorias e que parte as entregue ao veículo coletor. Entretanto, no PRVCE, os itens entregues por um cliente podem ser reaproveitados por outro cliente. Assim, não há a obrigatoriedade de precedência da entrega em relação à coleta. A cada cliente são associados dois valores, m_i e p_i , que representam, respectivamente, a quantidade de produtos a ser distribuída e recolhida pelo veículo.

A cada cliente também associamos os valores O_i e D_i . O primeiro indica o cliente do qual o cliente i receberá sua mercadoria, enquanto o segundo valor indica o cliente de destino do produto entregue pelo cliente i . Assim, o cliente O_i deve ser atendido antes do cliente i que, por sua vez, deve preceder o cliente D_i . Naturalmente, é possível definir o depósito como o cliente O_i ou como o cliente D_i .

Neste problema, a restrição de capacidade é construída de modo que, em cada trecho da rota, a soma dos itens a serem entregues e a serem coletados não exceda a capacidade do veículo.

3. Problema de Roteamento de Veículos com Múltiplos Depósitos (PRVMD) - Multi-Depot Vehicle Routing Problem (MDVP)

Este problema se diferencia do problema capacitado pela existência de vários depósitos, cada qual abrigando uma frota de veículos. Entretanto, ao final de sua rota, um veículo deve sempre retornar ao seu depósito de origem, sem passar por outros depósitos.

O PRVMD consiste em construir um conjunto de rotas para cada depósito, de modo a minimizar o custo total do transporte, satisfazendo o mesmo tipo de restrições do PRVC. Naturalmente, para evitar a resolução do PRVMD, que é mais complexa, podemos agrupar os clientes próximos em conjuntos, e designar cada grupo de clientes a um depósito, gerando vários PRVC, um para cada depósito, que podem ser resolvidos separadamente. Como extensão a esse problema, é possível definir depósitos intermediários pelos quais um veículo pode passar para reabastecer, antes de finalizar a sua rota. Existem poucos trabalhos na literatura explorando esse problema que, em inglês, recebeu o nome de *Multi-Depot Vehicle Routing Problem with Inter-Depot Routes* (MDVRPIR). No MDVRPIR, definimos uma *rota* como a seqüência de clientes a serem visitados por um veículo entre dois depósitos quaisquer, e um *roteiro* como a seqüência de clientes e depósitos visitados por um veículo desde a sua partida até a volta ao seu depósito central. Assim, um roteiro contém várias rotas.

4. **Problema de Roteamento de Veículos com Múltiplo Uso de Veículos (PR-MUV) - Vehicle Routing Problem with Multiple Use of Vehicle (VRPMUV)**

Quando a frota de veículos é pequena, ou a duração média das rotas é bem menor que um dia, é necessário atribuir mais de uma rota a um veículo. Neste caso, temos o Problema de Roteamento com Múltiplo Uso de Veículos. O objetivo é o mesmo do problema capacitado, ou seja, a determinação das rotas que, somadas, forneçam o menor custo total. As restrições do PRVC também são utilizadas no PRMUV.

5. **Problema de Roteamento de Veículos com Frota Heterogênea (PRVFH) - Heterogeneous Fleet Vehicle Routing Problem (HFVRP)**

Este problema diferencia-se do problema capacitado pelo fato de cada veículo apresentar uma capacidade específica¹. Existem duas variantes para o problema com frota heterogênea. Na primeira, o número de veículos de cada tipo é pré-estabelecido, de modo que para resolver o problema basta atribuir uma rota para cada veículo. Na segunda, o

¹Também é possível definir características particulares de velocidade, de tempo de viagem e de custos variáveis e fixos para cada veículo.

número de veículos de cada tipo não é especificado. Assim, além de construir uma rota para cada veículo, é necessário determinar o número de veículos utilizados. A esse último problema dá-se o nome de Problema de Dimensionamento e Roteamento de uma Frota Heterogênea de Veículos (ou *Fleet Size and Mix Vehicle Routing Problem*, em inglês).

Em ambas as variantes, as restrições são similares às do PRVC, lembrando que cada veículo possui uma restrição de capacidade particular.

6. Problema de Roteamento de Veículos com Entregas Fracionadas (PRVEF) - Split Deliveries Vehicle Routing Problem (SDVRP)

O Problema de Roteamento de Veículos com Entregas Fracionadas foi introduzido na literatura recentemente. Nele, os clientes podem ser atendidos por mais de um veículo, contrariando uma restrição imposta ao problema capacitado. Desta forma, é preciso definir também uma nova restrição de capacidade, que deve garantir que a soma das frações de demanda dos clientes visitados por um dado veículo não exceda a capacidade deste.

7. Problema de roteamento de veículos com janelas de tempo (PRVJT) - Vehicle Routing Problem with Time Window (VRPTW)

Este problema é o objetivo de nosso estudo e está descrito na próxima seção.

2.3 Problema de Roteamento de Veículos com Janela de Tempo.

O Problema de roteamento de veículos com janela de tempo (PRVJT) é a generalização do problema capacitado na qual se associa, a cada cliente, um período de tempo no qual algum veículo deve começar a atendê-lo. A esse intervalo dá-se o nome de *janela de tempo*.

Como no PRVC, o problema é representado por um grafo $G = (V, A)$, em que A é o conjunto de arcos e $V = 0, \dots, n$ é o conjunto de vértices. O vértice 0 indica o depósito, enquanto os demais nós representam os clientes.

O conjunto de clientes tem sua demanda satisfeita por uma frota de veículos que partem do depósito. Neste trabalho, não exigiremos que a frota tenha tamanho fixo, mas apenas um limite superior K . Dessa maneira, o número de veículos será aquele necessário para minimizar o custo de distribuição das mercadorias. Todos os veículos possuem uma capacidade constante C , de forma que a frota é homogênea.

A cada cliente i associamos:

- uma demanda m_i ;
- um tempo de serviço s_i ;
- um instante de início da janela de tempo e_i ;
- um instante de término da janela de tempo l_i ;

O depósito também possui uma janela de tempo $[e_0, l_0]$, indicando o momento a partir do qual os veículos podem começar a trafegar e o momento em que o último veículo deve estar de volta.

Em virtude da existência de janelas de tempo, é necessário associar a cada arco (i, j) , além do custo $c_{i,j}$, o escalar $t_{i,j}$ que representa o tempo necessário para ir do vértice i ao vértice j . Por simplicidade, adotaremos $c(i, j) = t(i, j)$ em nossos testes, o que corresponde a atribuir o valor 1 à velocidade dos veículos. Além disso, supomos sempre que a desigualdade triangular é respeitada.

Embora existam problemas em que a janela de tempo pode ser violada mediante pagamento de penalidades, trabalharemos aqui com o caso em que isso não é permitido. Por outro lado, um veículo pode chegar ao endereço de um cliente j antes do início de sua janela de tempo. Neste caso, o veículo deve permanecer parado em espera até o início do serviço.

O objetivo do problema é a minimização do custo total de transporte, que é representado pela distância total percorrida. Trabalhamos com Kn^2 variáveis binárias e n variáveis reais. Assim como no PRVC, a variável binária x_{ijk} indica se o veículo k percorreu ou não o arco que liga o nó i ao nó j . Por sua vez, a variável real b_i , $i = 1, \dots, n$, indica o instante de início do atendimento ao cliente i .

A formulação matemática do problema é dada por

$$\text{Minimizar} \quad \sum_{i=0}^n \sum_{\substack{j=0 \\ j \neq i}}^n \sum_{k=1}^K c_{ij} x_{ijk} \quad (2.9)$$

$$\text{sujeito a} \quad \sum_{k=1}^K \sum_{j=1}^n x_{0jk} \leq K \quad (2.10)$$

$$\sum_{j=1}^n x_{0jk} = \sum_{j=1}^n x_{j0k} \leq 1, \quad k = 1, \dots, K \quad (2.11)$$

$$\sum_{k=1}^K \sum_{j=0}^n x_{ijk} = 1, \quad i = 1, \dots, n \quad (2.12)$$

$$\sum_{j=0}^n x_{ijk} - \sum_{j=0}^n x_{jik} = 0, \quad k = 1, \dots, K, \quad i = 1, \dots, n \quad (2.13)$$

$$\sum_{k=1}^K \sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - v(S), \quad \forall S \subseteq V \setminus \{0\}, \quad |S| \geq 2, \quad (2.14)$$

$$\sum_{i=1}^n m_i \sum_{\substack{j=0 \\ j \neq i}}^n x_{ijk} \leq C, \quad k = 1, \dots, K \quad (2.15)$$

$$\sum_{k=1}^K \sum_{\substack{i=0 \\ i \neq j}}^n x_{ijk} (b_i + s_i + t_{ij}) \leq b_j \quad j = 1, \dots, n \quad (2.16)$$

$$e_i \leq b_i \leq l_i \quad i = 0, \dots, n \quad (2.17)$$

$$x_{ijk} \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, n, \quad k = 1, \dots, K. \quad (2.18)$$

A função objetivo (2.9) e as restrições (2.10)-(2.15) e (2.18) foram herdadas do PRVC. A restrição (2.16) relaciona o instante de início do atendimento de dois clientes visitados consecutivamente por um mesmo veículo. Ela evita que o intervalo entre os instantes de início do atendimento desses clientes seja inferior à soma do tempo gasto no atendimento do primeiro cliente com o tempo consumido na viagem entre os dois clientes. Por sua vez, (2.17) impede que o início do atendimento do cliente i se dê fora de sua janela de tempo.

A folga da restrição (2.16) corresponde ao tempo de espera do veículo que atende o cliente j , ou seja, o tempo consumido entre o instante de chegada do veículo ao endereço do cliente e o início do atendimento deste.

Técnicas de Solução

O Problema de Roteamento com janela de tempo tem sido objeto de extensivas pesquisas, tanto no campo das heurísticas e meta-heurísticas quanto dos métodos exatos, o que acaba por oferecer-nos uma vasta literatura relacionada ao tema. Uma análise das primeiras técnicas de solução do problema foi feita por Golden e Assad [54, 55], por Desrochers *et al.* [36], e por Solomon e Desrosiers [102]. Alguns autores escreveram artigos comparando os métodos de solução para o problema, como fizeram Braÿsy e Gendreau [24, 25], que trabalharam com heurísticas e meta-heurísticas para o PRVJT. Já Braÿsy *et al* [23] fazem um estudo sobre os algoritmos evolucionários dedicados ao PRVJT. Neste capítulo descreveremos, de uma maneira sucinta, os principais métodos desenvolvidos até o momento para a solução deste problema.

3.1 Métodos Exatos.

Os principais métodos exatos utilizados na resolução de PRVJT são o *Branch-and-Bound* clássico, o *Branch-and-Cut* e o *Branch-and-Price*. Todos eles utilizam alguma técnica de relaxação do problema de programação inteira, combinada, dependendo do método, com o uso de planos de corte e de geração de colunas.

3.1.1 *Branch-and-Bound.*

O algoritmo *Branch-and-Bound* é um método para otimização global que faz uma enumeração inteligente dos pontos candidatos à solução ótima de um problema, sendo bastante utilizado para a solução de problemas de programação inteira.

Considerando o problema de roteamento com janela de tempo, ou seja, um problema de programação inteira mista 0–1, o método tem início pela solução do problema no qual todas as restrições de integralidade são relaxadas. Em seguida, a técnica é decomposta em duas partes. Na fase de *branching* (ou de ramificação), uma variável binária é fixada, ora em 0, ora em 1, gerando, assim, dois subproblemas que devem ser resolvidos separadamente. O valor da função objetivo da solução ótima de cada um desses subproblemas é um limitante inferior para a solução ótima do problema original de programação inteira.

Ao conjunto de subproblemas relaxados associamos uma árvore que tem como raiz o problema em que todas as restrições de integralidade são relaxadas. Cada mudança de nível desta árvore está associada à ramificação de alguma variável. Os subproblemas ainda não resolvidos formam os nós pendentes, ou seja, os ramos não explorados dessa árvore.

Sempre que a solução ótima de um subproblema, x_k^* , satisfaz todas as restrições de integralidade do problema original, dá-se início a fase de *bounding*, que consiste na eliminação dos nós pendentes da árvore de subproblemas que têm função objetivo com valor pior que aquele correspondente a x_k^* .

Esse método é empregado por Baker [7] para resolver um problema de roteamento com restrições de janela de tempo e apenas um veículo, e por Kolen *et al.* [72], que resolve o PRVJT.

3.1.2 *Branch-and-Cut.*

O método de *Branch-and-Cut* consiste na combinação do procedimento de planos de corte com a técnica de *Branch-and-Bound*. Deste modo, após a fase em que o um problema relaxado é resolvido, o método de planos de corte é aplicado com o propósito de reduzir ainda mais a região factível dos subproblemas derivados desse nó da árvore. Essa redução é obtida através da adição de restrições que geram cortes no politopo factível do problema.

Mais detalhes sobre a aplicação do método ao PRVJT podem ser encontrados em Bard, Kontorauvis e Yu [9].

3.1.3 *Branch-and-Price*.

O método *Branch-and-Price* é essencialmente um método *Branch-and-Bound* com geração de colunas. Em cada nó da árvore de busca encontrada pelo método *Branch-and-Bound*, utiliza-se o método de geração de colunas com o propósito de encontrar novas soluções viáveis.

O método de geração de colunas representa uma generalização da decomposição de Dantzig-Wolfe [34], na qual o problema original é dividido em duas partes: um problema principal e um subproblema. Como o conjunto de todas as soluções factíveis é muito grande, apenas uma pequena parte desse conjunto de soluções é considerada no problema principal. Em cada iteração principal, determina-se se existe uma rota que possa reduzir a distância total. Isto é conseguido resolvendo o subproblema. Se tal rota existe, sua coluna correspondente é adicionada ao modelo e o procedimento é repetido até que a solução ótima seja encontrada.

Esse método foi aplicado ao PRVJT por Desrochers *et al.* [37]. Entretanto, a solução ótima só foi obtida para uma pequena parcela dos problemas de roteamento testados.

Jepsen *et al.* [67] aplicaram um método do tipo *Branch-and-Cut and Price* ao problema de roteamento com janela de tempo. O procedimento consiste na divisão do problema em um problema principal e um secundário, como é feito no método de geração de colunas realizado no *Branch-and-Price*. Além disso, planos de corte são adicionados ao problema principal. Com esse algoritmo, foi possível resolver 10 dos 56 problemas propostos por Solomon [102], problemas estes que serão apresentados no Capítulo 6.

3.2 Heurísticas.

Um conjunto de problemas pertence à classe P se existe algum algoritmo que encontra a sua solução ótima em tempo polinomial. Já os englobados pela classe NP são problemas computáveis cujas soluções, até o momento, somente são obtidas em um tempo exponencial, ou seja, ainda não são conhecidos algoritmos de complexidade polinomial capazes de resolvê-los.

Um problema é NP-completo se pertence à classe NP e todos os outros problemas desta classe são redutíveis a ele em tempo polinomial. Assim, se encontrarmos um algoritmo que resolva um problema NP-completo em tempo polinomial, podemos resolver todos os outros com a mesma complexidade. Um problema é NP-difícil se existe um problema NP-completo que pode ser reduzido a ele em tempo polinomial. Assim, um problema NP-difícil é ao menos tão difícil de resolver quanto os problemas NP-completos. Por outro lado, problemas NP-difíceis não precisam pertencer à classe NP. Para maiores informações sobre teoria de complexidade computacional, vide, por exemplo, Garey e Johnson [46].

O PRVTW enquadra-se na classe dos problemas NP-Difíceis [75]. A sua alta complexidade fez dos métodos heurísticos e meta-heurísticos uma alternativa viável quando é preciso encontrar soluções boas em um tempo limitado. Nesta seção, apresentamos, de uma maneira sucinta, os mais importantes métodos heurísticos para o problema, para depois introduzir as principais meta-heurísticas na próxima seção.

Os algoritmos heurísticos são aqueles que exploram apenas uma pequena parte do espaço de soluções, fornecendo uma solução de boa qualidade, com um custo computacional baixo. As heurísticas para o PRVJT podem ser divididas em algoritmos de construção e de refinamento.

3.2.1 Heurísticas de construção.

As heurísticas de construção são aquelas que geram uma solução factível passo a passo, a partir de uma solução trivial, geralmente infactível. A geração da solução factível pode ser feita tanto de forma seqüencial, ou seja, construindo uma rota por vez, como em paralelo, construindo todas as rotas ao mesmo tempo. As principais heurísticas para o problema de roteamento de veículos são:

- **Economias (*savings*).**

Desenvolvida por Clark e Wright [29] para resolver um problema de roteamento com restrição de capacidade, essa heurística tem por finalidade conectar duas rotas factíveis formando uma única rota, também factível. A escolha das rotas que serão unidas é feita de maneira que a junção gere a rota com o menor custo.

Como exemplo, suponha que temos dois circuitos C_1 e C_2 . Esses dois circuitos podem ser agrupados em uma única rota se removermos, de cada um, uma aresta que incide no depósito, ligando em seguida os nós pendentes. Assim, se as arestas $(i, 0) \in C_1$ e $(0, j) \in C_2$ são retiradas, podemos gerar uma nova rota combinando:

- o caminho de C_1 que liga o depósito ao nó i ;
- a aresta (i, j) ;
- o caminho de C_2 que liga o nó j ao depósito;

Neste caso, a economia gerada pela junção das rotas é dada por

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}. \quad (3.1)$$

O algoritmo funciona da seguinte maneira.

1. Crie n rotas, cada qual ligando um cliente ao depósito.
2. Enquanto existir mais de uma rota,
 - 2.1. Para cada par de rotas, calcule, segundo (3.1), a economia obtida ao reuni-las.
 - 2.2. Ordene as economias de forma crescente.
 - 2.3. Agrupe os dois circuitos cuja junção forneça a maior economia, desde que a rota assim gerada continue factível.

Solomon [101] baseou-se no procedimento de Clark e Wright [29] a fim de criar uma de suas heurísticas. A diferença se encontra na forma de obter a economia, que nesse caso é dada por

$$s_{ij} = c_{0i} + c_{j0} - \mu c_{ij},$$

em que μ é um valor entre 0 e 1. Muitas outras heurísticas baseadas neste método de economias podem ser encontradas na literatura (vide, por exemplo, [42]).

- **Varredura (*sweep*):**

Essa heurística foi desenvolvida por Gillett e Miller [50] para um problema de roteamento com restrições sobre a capacidade dos veículos e sobre a distância total máxima que pode ser percorrida em uma rota.

A heurística é simples e baseia-se na posição dos clientes no plano cartesiano. O algoritmo é descrito a seguir.

1. Instale o depósito na origem e escolha uma semi-reta que defina o eixo polar (de um sistema de coordenadas polares).
2. A cada cliente i , associe as coordenadas polares (r_i, θ_i) que fornecem sua localização no plano.
3. Crie uma lista de clientes pendentes seguindo a ordem crescente (ou decrescente) do ângulo θ .
4. Defina o depósito como ponto de partida da primeira rota.
5. Enquanto a lista de clientes pendentes não estiver vazia.
 - 5.1. Se o próximo cliente da lista puder ser incluído na rota atual,
 - 5.1.1. Inclua o cliente na rota.
 - 5.2. Caso contrário (isto é, se a inclusão do cliente na rota torná-la infactível),
 - 5.2.1. Inicie uma nova rota tendo esse cliente como primeiro nó após o depósito.
6. Para cada grupo de clientes que formam uma rota,
 - 6.1. Resolva aproximadamente o problema do caixeiro viajante resultante para determinar a rota dentro do grupo.

Para adaptar esse algoritmo ao problema com janela de tempo, Solomon [101] sugere que se defina a rota de cada veículo usando uma heurística de construção. Entretanto, nesta fase, pode ser necessário retirar alguns clientes das rotas para manter a factibilidade (em função da janela de tempo). Se isso ocorrer, o processo de Gillett e Miller [50] é reiniciado usando apenas os clientes excluídos das rotas.

- **Vizinho mais próximo.**

Nessa heurística, partimos do depósito e adicionamos, a cada iteração, o cliente ainda não escalado que está mais próximo do último inserido na rota corrente, desde que o circuito resultante seja factível. Se não for possível incluir um novo cliente no circuito atual, uma nova rota é iniciada. Esse procedimento é executado até que todos os clientes pertençam a alguma rota.

A medida de proximidade entre os nós i e j é dada por meio de um custo \bar{c}_{ij} . Em seu artigo, Solomon [101] calcula o custo dos arcos do PRVJT com base em três parâmetros $p_1, p_2, p_3 \geq 0$ tais que $p_1 + p_2 + p_3 = 1$. Dados esses parâmetros, o custo associado ao arco (i, j) é definido por meio da fórmula

$$\bar{c}_{ij} = p_1 d_{ij} + p_2 t_{ij} + p_3 v_{ij},$$

onde

$$\begin{aligned} t_{ij} &= b_j - (b_i + s_i), \\ v_{ij} &= l_j - (b_i + s_i + t_{ij}) \end{aligned}$$

e d_{ij} é a distância entre os nós. Observa-se que t_{ij} é a diferença de tempo entre o término do serviço em i e o início do atendimento em j . Já v_{ij} quantifica a urgência do atendimento do cliente j , dada pela diferença entre o instante final da janela de tempo em j e o instante de chegada do veículo neste nó.

- **Inserção mais barata.**

A heurística de inserção mais barata consiste na construção de uma solução de forma sequencial, adotando o menor custo como critério de seleção dos clientes. Para isso, é preciso analisar o custo de inserção de um cliente ainda livre entre cada par de nós i e j pertencentes à rota que está sendo construída. Em nosso algoritmo genético, utilizamos essa heurística, adotando os custos propostos por Solomon [101]. Sendo assim, recomendamos ao leitor interessado em conhecê-la melhor que consulte a Seção 5.2.

Uma versão em paralelo da heurística de inserção mais barata proposta por Solomon [101] foi introduzida por Potvin e Rosseau [87] para resolver um problema de roteamento com janela de tempo.

Ioannou *et al.* [66] modificaram a heurística seqüencial de Solomon [101], criando novos critérios de seleção e de inserção de cada cliente. No algoritmo de Ioannou *et al.* [66], esses critérios são baseados na minimização da função gulosa desenvolvida por Atkinson [3], segundo a qual o cliente escolhido para ser inserido na rota deve minimizar o impacto nos clientes já pertencentes à rota, nos que ainda não foram inseridos e na janela de tempo do cliente que está sendo escolhido para fazer parte da rota.

Além das heurísticas mencionadas acima, Bramel e Simchi-Levi [15] propuseram um algoritmo de duas fases para a solução do problema de roteamento com janela de tempo. Na primeira fase, k clientes são escolhidos para dar início às k rotas. Nessa fase, usa-se um procedimento similar à relaxação Lagrangiana. Em seguida, uma heurística gulosa é empregada visando a inserção de um cliente em uma rota. Essa inserção é feita de modo a minimizar o custo de inclusão de cada um dos clientes disponíveis em cada uma das k rotas.

Em seu estudo sobre as principais heurísticas de construção e de refinamento, Bräysy e Gendreau [24] compararam a heurística de inserção de Solomon [101], o procedimento desenvolvido por Potvin e Rosseau [87] e o método introduzido por Bramel e Simchi-levi [15]. Os melhores resultados foram aqueles obtidos por esta última heurística.

3.2.2 Heurísticas de Refinamento.

As heurísticas de refinamento nada mais são que técnicas de busca local, tendo como objetivo melhorar uma dada solução por meio da exploração de sua vizinhança. Existem duas classes de estratégias desse tipo. Na primeira, denominada *first accept* (FA) ou *first best* (FB), o exame da vizinhança cessa assim que uma solução melhor que a atual é encontrada. Na segunda classe, denominada *Best Accept* (BA) ou *Global Best* (GB), o exame acaba somente quando toda a vizinhança é explorada.

Em grande parte dos artigos mais recentes sobre problemas de roteamento, heurísticas de refinamento são usadas para melhorar a solução inicial ou como uma pós-otimização. Muitos autores também aplicam uma heurística de refinamento em conjunto com outra heurística de construção, ou seja, depois de adicionar os clientes às rotas, um refinamento é feito para melhorar a solução. Um exemplo desse procedimento para o PRVJT pode ser encontrado em Bräysy [20]

e Russell [97].

Apresentamos abaixo as heurísticas de refinamento mais empregadas na literatura.

- **K-opt.**

Essa técnica foi desenvolvida por Lin [76] para o problema do caixeiro viajante. O k -opt é um procedimento intra-rota que consiste em remover k arestas de uma rota e, então, selecionar k novas arestas, dentre todas as possíveis, a fim de formar um novo circuito. Naturalmente, isto é feito desde que a nova rota mantenha a factibilidade e tenha um custo menor.

A Figura 3.1 mostra um exemplo do 2-opt, no qual duas arestas foram removidas e, depois de testadas todas as possibilidades de reconecção dos nós, duas outras arestas foram incluídas na rota.

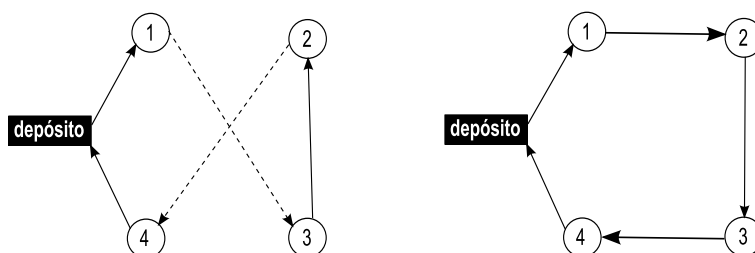


Figura 3.1: 2-opt. As arestas $(2, 4)$ e $(1, 3)$ foram substituídas por $(1, 2)$ e $(3, 4)$.

Naturalmente, a complexidade do método aumenta exponencialmente com o valor de k . Para $k \geq 3$, Lin e Kernighan [77] desenvolveram um método que executa uma seqüência finita de operações do tipo 2-opt. Um exemplo desta estratégia para o 3-opt é mostrado na Figura 3.2. Nesta figura, o desenho (A) simboliza a rota que será analisada, enquanto o grafo (B) mostra um procedimento 3-opt, na qual as arestas representadas por e_0 são substituídas pelas arestas que não pertencem ao grafo (A). O grafo (C) representa a execução de dois 2-opt consecutivos. No primeiro, as arestas representadas pela letra e_0 são substituídas pelas arestas (i, j) e (i^+, j^+) . No segundo 2-opt, as arestas representadas pela letra e_1 são removidas e são inseridas as arestas (i, k) e (j, k^+) . O resultado das operações efetuadas em (B) e em (C) é a rota apresentada na no grafo (D).

O procedimento desenvolvido por Lin e Kernighan consiste em retirar uma aresta e , então, reconectar a rota adicionando uma aresta adjacente àquela extraída. A outra aresta retirada deve ser adjacente a última inserida. Por fim, adiciona-se uma aresta adjacente a última que foi retirada. Assim, as quatro arestas envolvidas formam uma cadeia fechada, como pode ser observado no desenho (C) da Figura 3.2. O objetivo deste método é a geração de movimentos k -opt de boa qualidade, mas restringindo o espaço de busca, ou seja, limitando as escolhas das arestas removidas e adicionadas à rota.

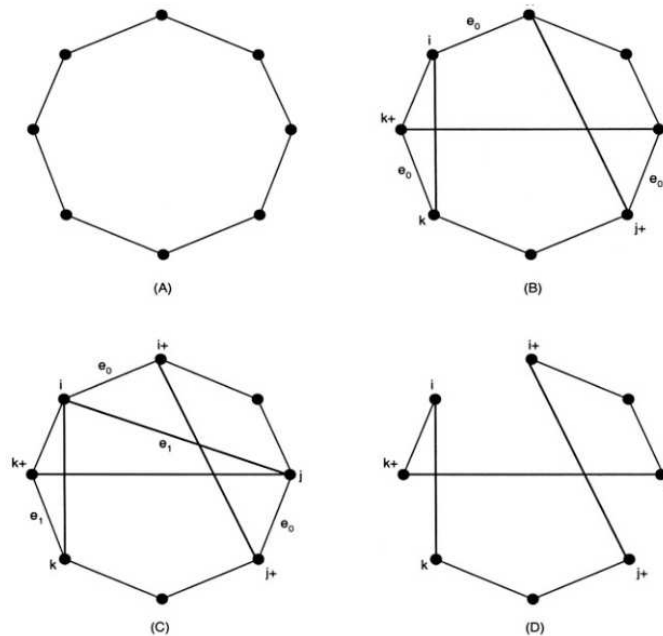


Figura 3.2: 3-opt obtido de 2 sucessivos 2-opt.

- **Or-opt**

Baseando-se na idéia de Lin [76], Or [84] também construiu um método intra-rota no qual uma cadeia de, no máximo, três vértices consecutivos escolhidos aleatoriamente é extraída da rota e, então, reinseridas entre dois nós do caminho restante, de modo que a rota continue factível e com uma solução melhor.

A Figura 3.3 mostra um exemplo do método. Neste exemplo, a cadeia com os nós 3 e 4 é extraída da rota por meio da remoção das arestas (2, 4) e (3, 5). Em seguida, esta cadeia é reinserida entre os nós 6 e 1 através da inclusão das arestas (3, 1) e (6, 4).

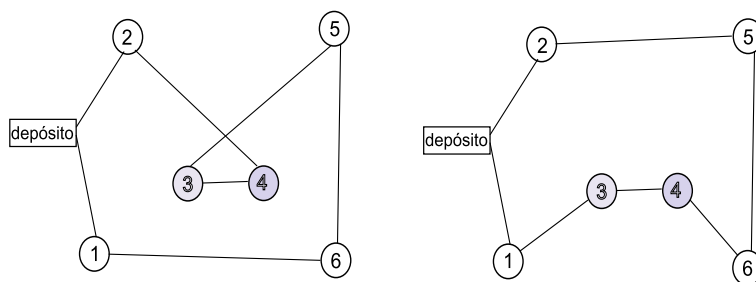


Figura 3.3: Or-opt.

- **2-opt***

Potvin e Rousseau [88] criaram o método 2-opt* adaptando o 2-opt de Lin [76] ao uso inter-rotas, além de ajustá-lo ao problema com janela de tempo, no qual a orientação da rota deve ser respeitada.

O 2-opt* tem início com a eliminação de duas arestas, cada qual pertencente a uma rota. Em seguida, duas novas arestas são inseridas, cada uma em uma rota, reconectando-as de modo a manter a factibilidade e a preservar a orientação.

A Figura 3.4 mostra um exemplo do método. O grafo da esquerda mostra a remoção das arestas (3,1) e (5,0). Como a orientação das rotas é preservada pelo 2-opt*, a única maneira de redefinir as duas rotas é através da inclusão das arestas (3,0) e (5,1).

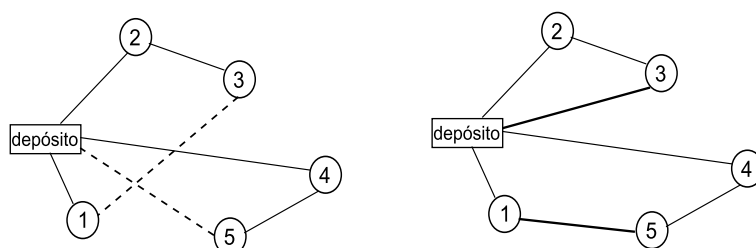


Figura 3.4: 2-opt*.

- **Troca- λ (λ -Interchange)**

Desenvolvido por Osman [86], o λ -Interchange é um método inter-rota. Dadas duas rotas distintas, em cada uma delas seleciona-se no máximo λ nós (a quantidade de nós pode diferir entre uma rota e outra). Em seguida troca-se os nós entre as rotas. As mudanças são aceitas somente se a solução continuar factível.

Um exemplo do 1-Interchange é apresentado na Figura 3.5. Como $\lambda = 1$, no máximo 1 cliente é escolhido em cada rota. Na figura, escolhemos, para a troca, os nós 3 e 4.

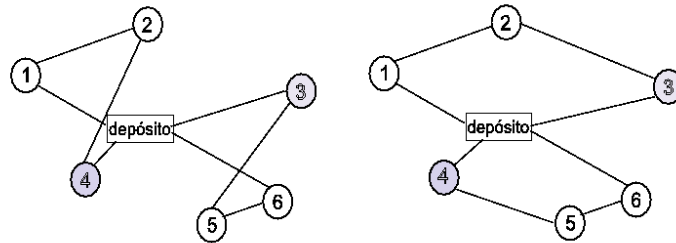


Figura 3.5: 1-Interchange.

Uma vez que empregamos este método em nosso algoritmo genético, sua idéia será melhor explorada no Capítulo 5.

- **Transferência cíclica**

Essa classe de métodos tem por objetivo melhorar o custo através da transferência de um determinado número de clientes entre várias rotas, de modo que os deslocamentos dos conjuntos de clientes ocorram de forma circular.

Suponhamos, então, que, dentre todas as rotas existentes, tenhamos selecionado apenas três, que denominaremos x , y e z . O procedimento consiste em tomar os clientes selecionados da rota x e inserí-los na rota y , ao mesmo tempo que os clientes escolhidos da rota y vão para a rota z e os clientes deste última rota são adicionados na rota x , concluindo o círculo. Se o número de clientes envolvidos em cada transferência for igual a k clientes, o método é denominado *transferência- k cíclica*. Já quando o subconjunto de rotas tem sua cardinalidade pré-definida, digamos b , o método é chamado de *transferência cíclica b* . Um estudo completo sobre esse método e suas variações é encontrado em Thompson e Orlin [110].

Para exemplificar o método, a Figura 3.6 apresenta uma transferência-2 cíclica-3, extraída de Thompson e Psaraftis [111], que investigaram a aplicação desse método a um problema de roteamento de veículos. No exemplo, temos quatro rotas I_1 , I_2 , I_3 e I_4 , das quais I_1 , I_2 e I_3 foram escolhidas. Em cada uma dessas rotas, dois nós foram selecionados para a

transferência. Da primeira rota, foram removidos os nós A_1 e A_3 , que foram adicionados à rota I_2 , em substituição aos nós B_1 e B_5 , acrescentados à rota I_3 . Por fim, os nós C_2 e C_3 deste última rota foram transferidos para a rota I_1 .

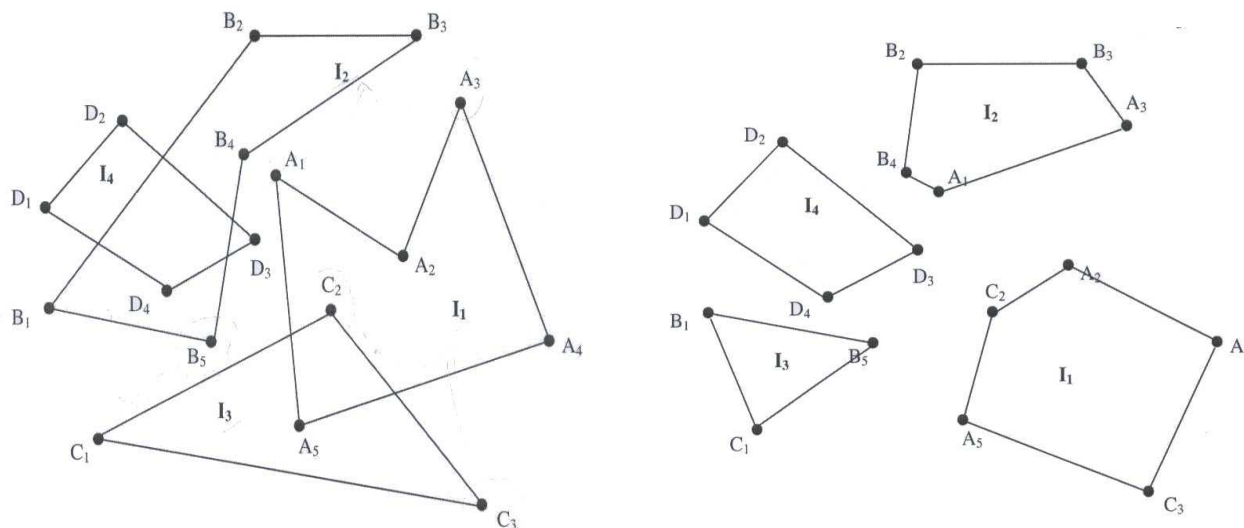


Figura 3.6: transferência-2 cíclica-3.

- **Cadeias de Ejeções (*Ejection Chain*)**

Este procedimento, baseado no método Lin e Kernighan [77], foi proposto para o problema do caixeiro viajante e envolve a geração de seqüências de movimentos nos quais as mudanças em determinados elementos fazem com que outros elementos sejam “ejetados” de suas posições. O ponto principal do algoritmo é a definição de *estruturas de referência*, que são usadas para coordenar a transformação eficiente de um subgrafo em outro pelo algoritmo de busca local. Uma descrição mais detalhada do método é dada por Glover [52] e [53].

Rego [92] define uma estrutura de flor como referência, e a usa para guiar a geração de caminhos e ciclos a fim de construir os vizinhos da solução corrente. Para isso, o grafo que representa uma solução é configurado de modo que o depósito passa a ser o núcleo de uma flor e as rotas representem as pétalas. O procedimento é arquitetado de modo que a estrutura de flor seja constituída por galhos e pétalas. Os galhos são obtidos removendo e adicionando arestas, de modo que a cadeia não contenha duas ligações com o depósito.

Em seu artigo, o autor propõe movimentos de ejeção que permitem a transição de uma estrutura de flor para outra. Esses movimentos cessam quando não existir mais opções de ejeção. A Figura 3.7 mostra a estrutura de flor empregada.

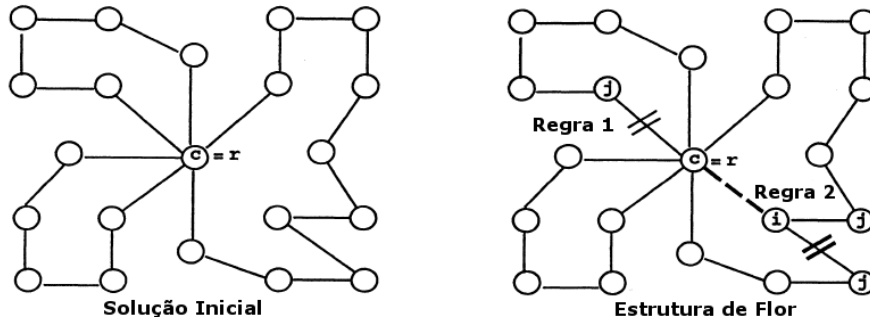


Figura 3.7: Cadeias de ejeções.

Uma aplicação direta das cadeias de ejeções ao PRJVT foi proposta por Sontrop *et al.* [103].

- **GENIUS (Generalized Insertion Procedure + Unstringing Stringing)**

Gendreau *et al.* [49] desenvolveram um procedimento de inserção (GENI) e uma rotina de pós-otimização (US) para o problema do Caixeiro Viajante. Da junção dos dois surgiu o GENIUS.

A principal característica do algoritmo de inserção baseia-se no fato de que um vértice pode ser adicionado entre dois nós que não necessariamente são consecutivos. Suponha, por exemplo, que se deseje inserir o vértice v entre os vértices v_i e v_j . No algoritmo GENI, essa operação pode ser feita de duas maneiras. A primeira, ilustrada na Figura 3.8, consiste em desconectar as arestas (v_i, v_{i+1}) , (v_j, v_{j+1}) e (v_k, v_{k+1}) , que são substituídas pelas arestas (v_i, v) , (v, v_j) , (v_{i+1}, v_k) e (v_{j+1}, v_{k+1}) . No segundo tipo, apresentado na Figura 3.9, as arestas (v_i, v_{i+1}) , (v_{l-1}, v_l) , (v_j, v_{j+1}) e (v_{k-1}, v_k) são substituídas por (v_i, v) , (v, v_j) , (v_l, v_{j+1}) , (v_{k-1}, v_{l-1}) e (v_{i+1}, v_k) . Observa-se que, em ambos os casos, a direção das arestas não é levada em conta. O procedimento de refinamento US consiste simplesmente em retirar um vértice da solução e então reconectá-lo em outra posição. Essa reinserção segue as duas estratégias descritas para o GENI.

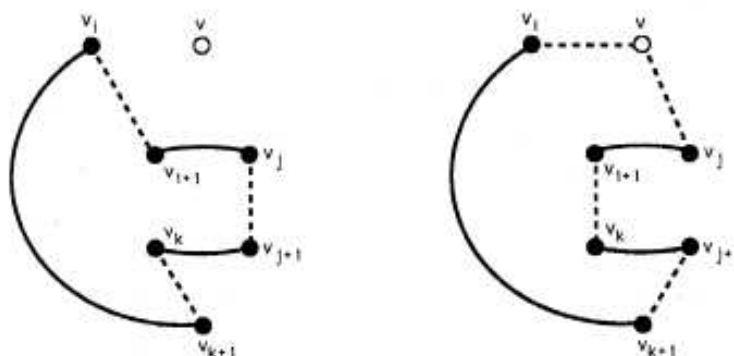


Figura 3.8: Geni I.

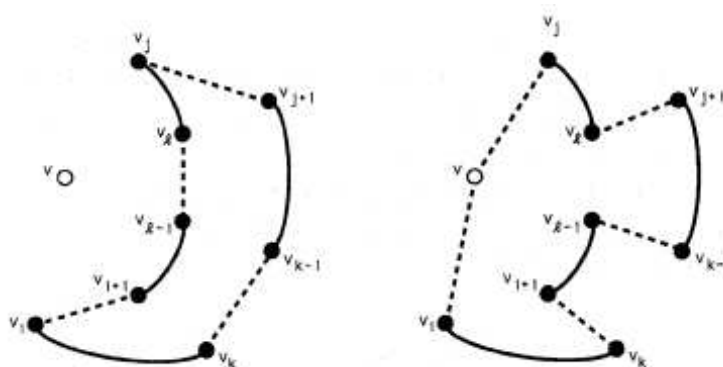


Figura 3.9: Geni II.

- **Troca Cruzada (*Cross-Exchange*)**

Esse método foi desenvolvido por Taillard *et al.* [104] para resolver um problema de roteamento com janela de tempo. Sua idéia central consiste em desconectar cadeias de nós, em duas rotas diferentes, removendo duas arestas de cada rota. Em seguida, as cadeias são permutadas entre as rotas. Como exemplo, a Figura 3.10 mostra a remoção das arestas (X_1, X'_1) e (Y_1, Y'_1) da primeira rota e a remoção de (X_2, X'_2) e (Y_2, Y'_2) da segunda rota. Com isso, os segmentos (X'_1, Y_1) e (X'_2, Y_2) , que contêm um número arbitrário de clientes, têm suas posições invertidas pela introdução das arestas (X_1, X'_2) , (X_2, X'_1) , (Y_1, Y'_2) e (Y_2, Y'_1) .

Algumas modificações desse método foram propostas por Braysy *et al.* [23]. Nesta nova versão da heurística, além de ser possível inserir uma cadeia no sentido inverso, também se

determina o melhor lugar dentro da rota para a inserção, em lugar de inserir uma cadeia na posição da qual a outra foi retirada.

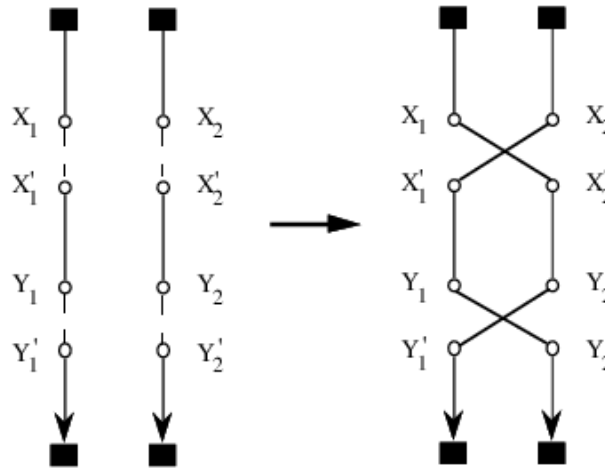


Figura 3.10: Cross-Exchange.

- **Busca em uma Vizinhança Grande (*Large Neighborhood Search*)**

Essa heurística, cuja abreviatura é LNS, foi introduzida por Shaw [100] para a resolução de um problema de roteamento. O método é baseado no processo contínuo de relaxação e reotimização, que consiste na retirada de alguns clientes pertencentes a uma rota e na reinserção desses clientes de modo a melhorar a solução. As variações no método são dadas pelas diferentes técnicas usadas na remoção e na reinserção dos clientes.

O critério de escolha dos clientes a serem removidos deve oferecer boas oportunidades de melhora na solução. Desse modo, supondo que um cliente já tenha sido removido, o autor sugere a remoção de outros clientes próximos àquele escolhido inicialmente. Para o problema com janela de tempo, Rousseau *et al.* [94] sugerem também a remoção dos clientes que antecedem e sucedem, na rota, os escolhidos segundo Shaw [100].

Para reinserir os clientes selecionados, Shaw [100] utiliza uma busca em árvore baseada no método *branch-and-bound*. Já Rousseau *et al.* fazem uma busca na árvore produzida pelo *branch-and-cut*.

3.3 Metaheurísticas.

Assim como as heurísticas, as meta-heurísticas têm como objetivo explorar apenas parte do espaço solução. Entretanto, geralmente, isto é feito de um modo mais abrangente, fazendo com que as soluções encontradas sejam de melhor qualidade. Contrariamente às heurísticas, as meta-heurísticas são mais gerais e têm capacidade de sair de ótimos locais. As mais importantes meta-heurísticas para o problema de roteamento são:

- **Busca Tabu (*Tabu Search*)**

Essa meta-heurística foi introduzida por Glover [51], mas foi Garcia *et al.* [45] que propuseram a primeira aplicação para o problema de roteamento e programação de veículos com janela de tempo. O conceito básico da busca tabu (BT) é explorar o espaço solução, a cada iteração, movendo de uma dada solução para outra que pertença à sua vizinhança. Diferentemente dos métodos clássicos de descida, aceita-se soluções piores, o que pode gerar ciclos. Para evitar a ciclagem, as soluções já avaliadas são marcadas como proibidas e incluídas em uma *lista tabu*.

Se, por um lado, a lista tabu impede a ciclagem, por outro pode proibir movimentos para soluções ainda não exploradas. Assim, emprega-se também um critério de aspiração, que permite que determinadas soluções saiam da lista tabu sobre certas circunstâncias.

A maioria dos artigos, nos quais a busca tabu é usada para resolver o PRVJT tem como objetivo principal reduzir o número de rotas. A construção da solução inicial e a construção dos vizinhos é feita, na maioria dos casos, empregando as heurísticas descritas nas seções acima, ou alguma modificação delas. O uso de estratégias de intensificação, que têm por objetivo concentrar a busca em determinadas regiões promissoras, e de diversificação, que utilizam a memória de longo prazo a fim de redirecionar a busca para regiões ainda não suficientemente exploradas do espaço solução, são comuns na BT. Rochat e Taillard [93] propuseram o emprego de uma memória adaptativa no desenvolvimento dessas estratégias para o problema de roteamento, o que pode ser facilmente estendido para o problema com janela de tempo.

Badeau *et al.* [6] e Schulze e Fahle [98] desenvolveram uma versão em paralelo da BT para

resolver um PRVJT. Lau *et al.* [74] propuseram uma Busca Tabu caracterizada por uma lista de espera para o problema em que o número de veículos é fixado e a janela de tempo pode ser violada através de penalizações. A lista de espera é constituída por clientes ainda não pertencentes a alguma rota. Operações de recolocação e trocas são executadas entre a solução parcial e a lista, como se esta fosse uma rota. A janela de tempo é relaxada para que se possa incluir mais clientes nas rotas.

- **Têmpera Simulada (*Simulated Annealing*)**

A têmpera simulada (TS) é uma técnica de relaxação estocástica baseada no processo térmico utilizado na metalurgia para obtenção de estados de baixa energia num sólido.

No algoritmo da TS, uma nova solução x_t é aceita sempre que $f(x_t) < f(x)$, onde x é a solução corrente. Para fugir dos mínimos locais, soluções com $f(x_t) \geq f(x)$ também são aceitas com uma probabilidade $e^{\delta/T}$, onde $\delta = f(x_t) - f(x)$ e T é um parâmetro (chamado temperatura) que varia ao longo das iterações, partindo de um número grande e terminando próximo de zero. A queda na temperatura ocorre gradativamente e costuma ser feita através da regra $T_k = \alpha T_{k-1}$ para $0 \leq \alpha \leq 1$.

Esse método foi introduzido por Metropolis *et al.* [79] e adaptado aos problemas de otimização por Kirkpatrick *et al.* [71]. A primeira aplicação para um PRVJT foi feita por Chiang and Russell [27]. Para mais informações veja Aarts *et al.* [1].

Uma modificação ao TS foi feita por Dueck and Scheurer [40], originando a heurística de refinamento *Threshold Accepting* (TA). A alteração consiste na eliminação do elemento estocástico, que é substituído por um termo determinístico \tilde{T} , conhecido como limiar (ou *threshold*). Assim, aceita-se uma solução x_t sempre que $f(x_t) < f(x) + \tilde{T}$. Como no SA, o parâmetro \tilde{T} é reduzido lentamente ao longo das iterações. Esse método se mostrou mais eficiente que a têmpera simulada para os problemas de roteamento. Braysy *et al.* [21] foram os pioneiros em aplicar o TA a um problema de roteamento com janela de tempo. Em [23], Bräysy *et al.* utilizam-no como um algoritmo de pós-otimização na segunda fase do seu algoritmo de 2 fases.

- **Algoritmo de Colônia de Formigas (*Ant Algorithms*)**

O método da colônia de formigas (CA) é inspirado na estratégia adotada pelas formigas para obter alimento. Em sua busca por comida, as formigas marcam seu caminho liberando uma substância denominada feromônio. Essa substância influencia o comportamento das outras formigas, que dão uma certa preferência às trilhas que contêm a substância.

As formigas que encontram os menores caminhos até a fonte de alimento costumam voltar mais rápido ao formigueiro, de modo que suas trilhas têm um pouco mais de feromônio que as demais, sendo mais visitadas por outras formigas. Além disso, o feromônio também evapora, de modo que caminhos pouco visitados perdem a substância e são descartados pelas formigas. Assim, ao longo do tempo, os menores caminhos recebem uma carga maior de feromônio e são mais utilizados.

No algoritmo CA, formigas artificiais trabalham cooperativamente, comunicando-se apenas através do feromônio deixado pelo caminho. Cada formiga constrói, sozinha, a cada iteração do algoritmo, uma solução para o problema de roteamento. A formiga tem uma memória que a permite reconstruir o caminho percorrido. Quando está em um nó r do problema, a formiga pode mover-se para qualquer nó s de sua vizinhança $N(r)$, desde que este ainda não tenha sido visitado. A escolha do próximo nó é feita com base em duas informações diferentes: a informação heurística, η_{rs} , que mede o impacto na função objetivo de se atravessar a aresta (r, s) , e a informação vinda do feromônio artificial, τ_{rs} , que mede quão desejável é a aresta (r, s) . A quantidade de feromônio pode ser atualizada à medida em que a formiga se move, ou ao final da iteração.

Coloni, Dorigo e Maniezzo [30] aplicaram o método para resolver um problema do caixeiro viajante. Bullnheimer *et al.* [26] utilizaram-no para o PRV. Já o PRVJT foi resolvido por Gambardella, Taillard e Agazzi [44], que utilizam duas colônias, a primeira para minimizar o número de veículos e a segunda para melhorar a distância percorrida.

- **Algoritmos Evolucionários (*Evolutionary Algorithms*)**

Os algoritmos evolucionários (AE) são métodos baseados nos mecanismos de seleção natural de Darwin. Esses algoritmos trabalham com uma população de soluções e buscam o

ótimo através de procedimentos de cruzamento, mutação e seleção de soluções pertencentes à população. Algoritmos Genéticos (AG), Estratégias Evolucionárias e Programação Evolucionária fazem parte dessa classe de técnicas. O algoritmo desenvolvido nesse trabalho se baseia nos Algoritmos Genéticos, de modo que suas idéias serão melhor exploradas no próximo capítulo.

Exemplos do uso de AE para resolver um PRVJT podem ser vistos em Homberger e Gehring [63], que propuseram duas estratégias evolucionárias diferentes. As duas estratégias utilizam uma aproximação estocástica baseada na heurística das economias, ou seja, os clientes pertencentes a lista das economias são escolhidos aleatoriamente para constituir a rota. A função objetivo pondera o número de rotas, a distância total e um critério que determina a facilidade de eliminação da menor rota. A mutação é feita pelas heurísticas de refinamento Or-opt, 2-opt* e 1-Interchange. No primeiro algoritmo, o cruzamento não é executado. No segundo, o cruzamento é feito através de um procedimento uniforme.

Jung e Moon [68] desenvolveram um algoritmo genético híbrido, no qual a função objetivo é baseada na distância. O algoritmo começa com a aplicação da heurística de inserção de Solomon [101] para a determinação de uma solução inicial. O primeiro cliente de cada rota é escolhido de forma aleatória entre o cliente mais distante do depósito, o cliente com o menor instante final da janela de tempo e um cliente também determinado aleatoriamente.

No algoritmo de Jung e Moon [68], a seleção é feita pelo torneio. Para o cruzamento, o grafo que contém o depósito, os clientes e as arestas utilizadas para formar as rotas de cada veículo são mapeados e a escolha dos pontos de corte é feita por meio de curvas ou figuras geométricas de diferentes tipos. A Figura 3.11, extraída de [68], exemplifica o procedimento de cruzamento. As regiões formadas pela sobreposição das figuras geométricas definem conjuntos de nós. Cada conjunto pertence a um único pai. Primeiramente, definimos as rotas dentro desses conjuntos. Como essa divisão de nós é arbitrária, restarão várias rotas desconexas, de modo que é preciso utilizar algoritmos de reparação para reconstruir uma solução factível. Essa reconstrução é feita seguindo a regra do vizinho mais próximo.

Na mutação, Jung e Moon [68] fazem mudanças de nós entre, no máximo, 3 rotas. As

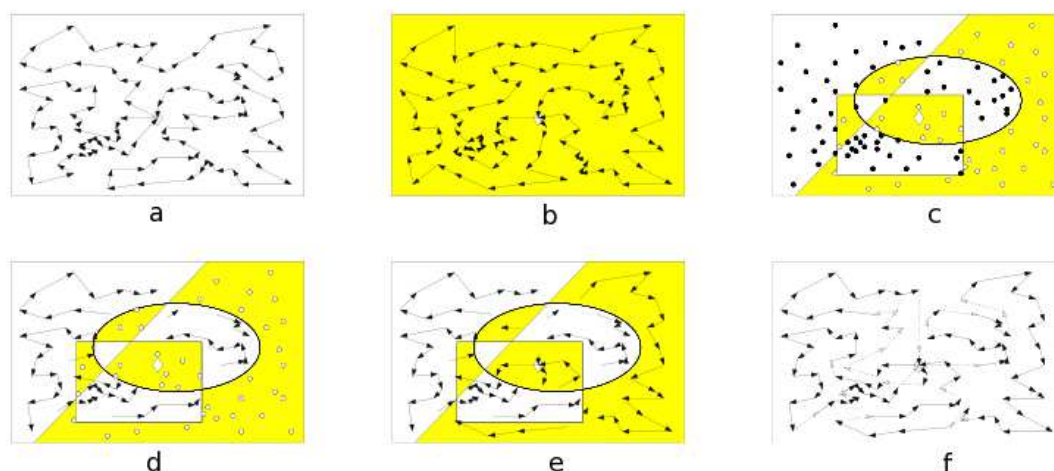


Figura 3.11: O crossover utilizado por Jung e Moon. As Figuras **a** e **b** representam, respectivamente, os pais 1 e 2. A Figura **c** mostra a divisão dos clientes com base em figuras geométricas. A Figura **d** mostra as ligações feitas nas regiões referentes ao primeiro pai, enquanto a Figura **e** mostra as rotas internas à região referente ao segundo pai. Finalmente, a Figura **f** mostra as rotas após a aplicação do algoritmo de reparação.

heurísticas de refinamento *or-opt*, *relocate* e *crossover*¹ são aplicadas ao final da iteração para melhorar a solução.

Algoritmos evolucionários para o problema com janela de tempo foram analisados e comparados por Bräysy *et al.* [22].

- **Busca Local com Múltiplos Pontos Iniciais (*Multi-Start Local Search*)**

Este método de busca local envolve a geração de um conjunto de soluções iniciais, seguida da aplicação de um procedimento de refinamento a cada solução gerada. As diferentes soluções iniciais permitem uma diversificação do espaço de busca, o que evita ótimos locais.

Exemplos da aplicação dessa meta-heurística ao PRVJT podem ser encontrados em Bräysy *et al.* [21, 23]. Nestes artigos, a heurística de inserção mais barata é usada para criar a solução inicial, que é refinada por uma extensão da heurística de cadeia de ejeções, com o

¹Heurística que promove mudanças entre nós de maneira a retirar arestas cruzadas. Não deve ser confundida com o procedimento de cruzamento adotado nos algoritmos genéticos.

objetivo de reduzir o número de veículos. Finalmente, uma modificação da troca cruzada é empregada para reduzir a distância total percorrida em cada solução.

- **GRASP (*Greedy Randomized Adaptive Search Procedure*)**

O GRASP (cuja tradução fiel seria “procedimento de busca adaptativo aleatorizado guloso”) é uma meta-heurística proposta por Feo e Resende [41], na qual cada iteração é dividida em uma fase de construção e uma busca local. O objetivo da fase de construção é a geração de uma solução factível. Para tanto, os clientes ainda não inseridos em uma rota são ordenados por uma função gulosa e uma porcentagem dos clientes melhor avaliados é selecionada para formar uma lista restrita de candidatos. Em seguida, um nó pertencente a essa lista é escolhido aleatoriamente e adicionado à rota que está sendo construída. Este procedimento é repetido até que todos os clientes tenham sido selecionados. Na segunda fase, uma busca local é executada a fim de encontrar um ótimo local próximo à solução gerada na fase de construção. Esse método foi adaptado ao PRVJT por Kontoravdis e Bard [73].

- **Busca em Vizinhança Variável (*Variable Neighborhood Search*)**

A busca em vizinhança variável (VNS) é um método de busca local que consiste em explorar o espaço solução através de mudanças sistemáticas de estruturas de vizinhança. Ou seja, dada a solução corrente r e uma lista ordenada de estruturas de vizinhança $\{N_1(r), N_2(r), \dots, N_m(r)\}$, a cada iteração, uma solução \bar{r} pertencente a uma vizinhança $N_i(r)$ é escolhida. Em seguida, uma busca local é aplicada a este vizinho selecionado, gerando uma nova solução r^* . Se r^* for melhor que r , ela assume o lugar da solução corrente e volta-se à primeira estrutura de vizinhança, $N_1(r^*)$. Caso contrário, apenas transfere-se a busca à próxima estrutura de vizinhança, $N_{i+1}(r)$.

Esse método foi introduzido por Mladenović e Hansen [81]. Uma versão simplificada do método, proposta pelos mesmos autores, é chamada Método de Descida em Vizinhança Variável (*Variable Neighborhood Descent*, ou VND). Neste método, utiliza-se a estratégia *global best* para se obter o vizinho dentro de uma estrutura de vizinhança, abandonando-se a busca local. Segundo os autores, o VND se apóia no fato de que um ótimo local

obtido para um estrutura de vizinhança não corresponde, necessariamente, ao ótimo de outra estrutura de vizinhança. Entretanto, um ótimo global corresponde ao ótimo local de todas as estruturas de vizinhança.

Bräysy [20] construiu um método de quatro fases que emprega o VND para resolver um PRVJT. Neste método, depois de uma solução ser criada e refinada, o VND é aplicado com intuito de melhorar a distância total percorrida na solução. As estruturas de vizinhança usadas pelo autor são denominadas *Icross*, *Insert Related Parallel* (IRP), IOPT e O-opt. A primeira se baseia na troca cruzada, enquanto a segunda é derivada do LNS. As duas últimas estruturas são derivadas do Or-opt.

- **Busca Local Guiada (*Guided Local Search*)**

A Busca Local Guiada (GLS), desenvolvida por Voudouris [115] e por Voudouris e Tsang [116], foi aplicada ao problema com janela de tempo por Kilby *et al.* [70]. O GLS possui alguma similaridade com a Busca Tabu. O método tenta evitar mínimos locais adicionando penalizações na função objetivo, com base na experiência adquirida a partir da busca feita ao longo das iterações. O método opera penalizando soluções com características particulares, sendo cada característica associada a um custo e à quantidade de vezes em que foi penalizada.

- **Metaheurísticas Híbridas**

A maioria dos métodos de solução desenvolvidos nos últimos anos usa as idéias mencionadas nessa seção de uma forma mista, ou seja, mescla meta-heurísticas, ou uma destas com algum tipo de heurística. Essa mistura tem fornecido resultados melhores que os obtidos pelos métodos tradicionais. Alguns artigos que abordam esse tipo de técnica para resolver o PRVJT estão listados abaixo.

Thangiah *et al.* [106] desenvolveram um método em duas fases. Na primeira, uma solução inicial é criada usando a heurística de inserção de Solomon [101] e um algoritmo genético baseado em setores. Na segunda fase, usa-se o λ -Interchange combinado com TS e BT.

Homberger e Gehring [47] desenvolveram outro método de duas fases. Na primeira, procura-se minimizar o número de rotas usando a estratégia evolucionária desenvolvida

por Homberger e Gehring [63]. Na segunda fase, uma busca tabu é empregada com o intuito de minimizar a distância total. Gehring e Homberger [48] introduziram melhorias no algoritmo de Gehring e Homberger [63], incluindo novos critérios de parada e o uso da folga do caminhão como critério de eliminação de uma rota.

Bräysy *et al.* [19] também utilizam um método de duas fases. Na primeira, um algoritmo genético é empregado com a finalidade de obter uma solução factível. Uma outra estratégia evolucionária se encarrega de melhorar a solução na segunda fase.

Bräysy [20] propôs um algoritmo com quatro fases. Na primeira, constrói-se a solução inicial por meio de uma heurística de inserção mais barata. Após a inserção de k clientes na solução o procedimento Or-Opt é executado. Na segunda fase, emprega-se uma adaptação do *ejection chain* com o intuito de reduzir o número de veículos. Na terceira e quarta fases, usa-se método de descida em vizinhança variável (VND) com a finalidade de minimizar a distância total percorrida. Um vez que esse método apresentou um desempenho excelente, iremos empregá-lo, no Capítulo 5, para avaliar o algoritmo genético que propomos neste trabalho.

Algoritmos Genéticos

Através de observações da natureza, percebeu-se que ela resolvia seus problemas, em muitos dos casos de alta complexidade, de forma elegante e eficiente. Darwin observou que as espécies se adaptavam ao ambiente no qual viviam. Indivíduos mais capazes de sobreviver neste ambiente eram naturalmente selecionados e tinham maior probabilidade de procriação. Consequentemente, eram estes os que mais passavam adiante suas características, gerando um ciclo evolutivo seletivo.

Naquela época, sua teoria foi bastante criticada, pois não era conhecido o mecanismo segundo o qual as características eram passadas de pai para filho. Entre 1936 e 1947, as contribuições da genética (descobertas de Mendel), permitiram o conhecimento dos genes e da mutação, o que explicou o processo evolutivo e a variedade das espécies.

Neste capítulo, abordaremos os Algoritmos Genéticos, derivados dessa teoria da evolução natural. O estudo se inicia com a introdução dos Algoritmos Evolucionários, grupo de métodos ao qual os algoritmos genéticos pertencem. Em seguida, apresentaremos os conceitos envolvidos nessa técnica, além de ilustrar uma estrutura básica do algoritmo. Por fim, abordaremos a teoria de convergência do método.

4.1 Algoritmos Evolucionários.

Os Algoritmos Evolucionários (AE) são uma classe de métodos de busca que imitam o processo de evolução natural. Eles trabalham com uma população de soluções a cada iteração, ao invés de utilizar uma única solução, como é usual em meta-heurísticas. Darwin indentificou três princípios básicos da evolução: reprodução, seleção natural e diversidade dos indivíduos. É com base nesses princípios que essa classe de técnicas foi desenvolvida, modelando os processos de seleção, reprodução e mutação. Segundo Darwin, os indivíduos com melhores características, ou seja, os mais adaptados, tendem a sobreviver frente aos demais. Em um algoritmo evolucionário, atribui-se a cada indivíduo uma pontuação associada à sua adaptação ao problema. Aos mais adaptados é dada a oportunidade de reproduzir-se mediante cruzamentos com outros indivíduos da população. O processo de evolução é aleatório, mas direcionado, pois explora informações históricas para encontrar novos pontos de melhor desempenho.

Diferentes escolas dedicaram-se aos AE nestes últimos 40 anos. Os principais métodos desenvolvidos nessa linha incluem os Algoritmos Genéticos (AG), criados nos Estados Unidos por Holland em conjunto com seus alunos e colegas da Universidade de Michigan [62], as Estratégias Evolucionárias, desenvolvidas na Alemanha por Rechenberg [90] e Schwefel [99], e a Programação Evolucionária de Lawrence J. Fogel [43]. As principais diferenças entre eles são a forma com que o indivíduo é representado e o papel da mutação dentro do processo evolutivo.

Neste trabalho, nos concentraremos nos algoritmos genéticos.

4.2 Características Gerais dos Algoritmos Genéticos.

O algoritmo genético é estruturado de forma que as informações referentes a cada candidato a solução são codificadas similarmente aos cromossomos biológicos. Um vetor solução do problema é denominado cromossomo. Por sua vez, cada elemento do vetor solução, também chamado de gene, pode ser representado de uma forma binária, inteira ou real. Ao conjunto de vários cromossomos (ou indivíduos) dá-se o nome de população.

A cada iteração do algoritmo, também chamada geração, um novo conjunto de indivíduos é gerado através de trocas de informações (genes) entre os cromossomos da geração anterior.

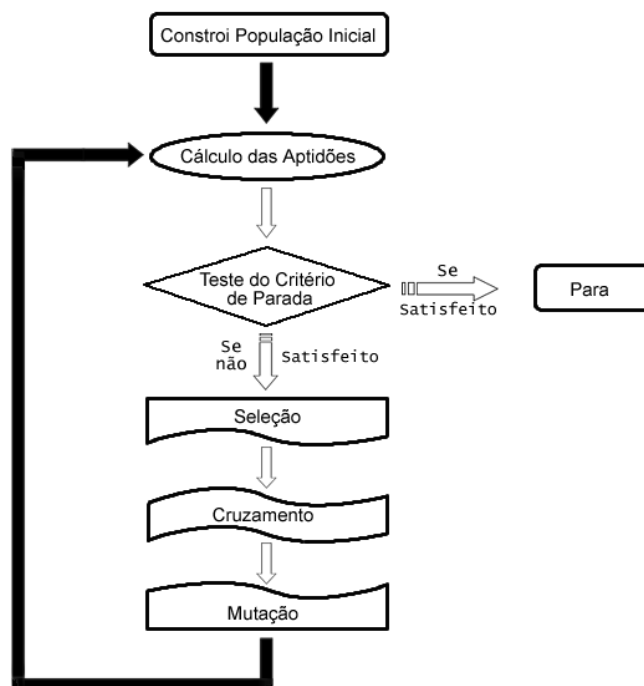


Figura 4.1: Esquema de um algoritmo genético básico.

O resultado dessas trocas é o aumento da adaptação dos indivíduos ao meio, o que acarreta em uma melhora global da população, fazendo com que, a cada geração, nos aproximemos da solução ótima. Os processos que mais contribuem para a evolução são o cruzamento entre indivíduos, a adaptação baseada na seleção e a mutação.

A estrutura básica de um AG simples segue o esquema ilustrado na Figura 4.1.

A aptidão do indivíduo é determinada através da função objetivo. O algoritmo chega ao fim quando os critérios de parada, determinados *a priori* e dependentes do problema, são satisfeitos.

4.2.1 Operadores Genéticos.

Responsáveis pela transformação da população nas sucessivas gerações, os operadores genéticos têm por objetivo manter a diversidade da população e, ao mesmo tempo, transmitir as características adquiridas nas gerações anteriores, provocando uma melhora da aptidão ao longo do processo.

- **Seleção:** tem como objetivo fazer com que os indivíduos que apresentam os melhores valores da função objetivo passem para a próxima geração. As estratégias de seleção mais utilizadas são o método da roleta e a seleção por torneio. Essas estratégias serão apresentadas no próximo capítulo.
- **Cruzamento** (*Crossover*): é a troca de segmentos de genes entre pares de cromossomos selecionados, com o objetivo de gerar novos indivíduos. Sua finalidade é a combinação das características dos indivíduos da população. Formas comuns de cruzamento são aquelas em que há 1 ponto ou 2 pontos de cruzamento entre os genes.
- **Mutação:** é a alteração arbitrária de um ou mais genes de um cromossomo escolhido. É necessária para a introdução e a manutenção da diversidade genética da população, o que evita que o AG convirja muito cedo para mínimos locais.

4.2.2 Parâmetros Genéticos.

Além da forma como o cromossomo é codificado, existem vários parâmetros do AG que podem ser escolhidos para melhorar seu desempenho, adaptando-o às características particulares de determinadas classes de problemas. Entre eles, os mais importantes são:

- **Tamanho da População:** quanto maior a população, mais ampla será a cobertura do espaço solução do problema, permitindo, assim, a prevenção de uma convergência prematura. No entanto, uma população muito grande exige maiores recursos computacionais e, conseqüentemente, um período de tempo maior para a obtenção da solução.
- **Probabilidade de Cruzamento:** quanto mais alta a probabilidade de cruzamento, mais rapidamente novas estruturas serão introduzidas na população. Entretanto, se for muito alto, este parâmetro pode destruir boas soluções. Por outro lado, se for muito baixo, pode tornar lenta a convergência.
- **Probabilidade de Mutação:** sua principal função é promover a diversidade da população. Assim, este parâmetro deve ser ajustado de forma a prevenir uma convergência

prematura para mínimos locais. Geralmente se utiliza uma taxa de mutação pequena, para evitar que as boas características da população sejam destruídas.

- **Número de Gerações:** exerce influência no desempenho do algoritmo. Um número grande de iterações torna o método lento, enquanto um número pequeno pode impedir que uma boa solução seja obtida. O AG é uma busca estocástica, e não é possível garantir que a solução encontrada após um certo número de iterações seja ótima. Na maioria dos artigos publicados na literatura, o número de iterações é previamente escolhido. Entretanto, como veremos ao final deste capítulo, estudos recentes permitem determinar o menor número de iterações necessárias para obter uma solução ótima com uma probabilidade predefinida.
- **Pressão Seletiva:** o conceito de pressão seletiva representa a influência do meio ambiente no processo de seleção dos genes ao longo das iterações. Quando os cromossomos da população possuem valores de aptidão similares, a probabilidade de sobrevivência dos indivíduos também é semelhante, caracterizando um ambiente com pouca pressão seletiva. Em contrapartida, quando os valores de aptidão são bem distintos, a probabilidade de bons indivíduos serem selecionados é maior, caracterizando uma alta pressão seletiva. O ajuste da pressão seletiva permite induzir ao AG a uma convergência mais ou menos rápida a uma solução, ainda que não necessariamente a melhor. Portanto, seu controle é de extrema importância para que o processo evolutivo seja efetivo.

A influência de cada parâmetro no desempenho do algoritmo depende da classe de problemas com que se está trabalhando. Assim, a determinação de um conjunto de valores otimizados para estes parâmetros depende da realização de um grande número de testes, como veremos no Capítulo 6.

4.2.3 Estrutura do Algoritmo Genético.

Devido ao grande interesse no estudo dos AG, muito se evoluiu desde seu surgimento, principalmente em relação a sua estrutura. Nessa seção iremos apresentar detalhadamente os passos básicos de um algoritmo genético simples, também chamado de canônico, conforme o exposto por Michalewicz [80]. Variações dessa estrutura básica serão descritas no próximo capítulo.

- **Representação dos Indivíduos**

No algoritmo genético canônico, os cromossomos são codificados de modo que cada solução real, x , é representada por um vetor binário, v , como mostrado abaixo.

$$v = (110001011001010)$$

Cada componente do vetor corresponde a um gene. O comprimento do vetor dependerá da precisão exigida para a solução do problema a ser resolvido.

- **Função de Avaliação**

A aptidão do indivíduo é medida através de uma função de avaliação. Como essa função varia de acordo com o problema, usaremos a notação

$$eval(v) = f(x),$$

onde $f(x)$ fornece um valor real que indica o quão adaptado está o cromossomo ao meio.

- **População**

A população é um conjunto formado por n cromossomos (ou indivíduos).

Passos de um AG.

Os passos descritos abaixo se baseiam no Algoritmo Genético Clássico, representado na Figura 4.1. Construída uma população inicial com n indivíduos, cada qual com um valor de aptidão, a cada geração, uma nova população é gerada como resultado das aplicação dos procedimentos de seleção, cruzamento e mutação, sendo o cálculo da aptidão executado somente na etapa da seleção.

Passo 1: Seleção

O procedimento de seleção que apresentaremos aqui é muito simples e simula o processo de uma roleta. A área total da roleta, chamada de função de desempenho total, F , é definida como a soma dos valores de aptidão de todos os cromossomos da população.

$$F = \sum_{i=1}^n (eval(v_i)).$$

A área total é dividida em pedaços, cada qual correspondendo à probabilidade de um certo cromossomo ser escolhido. Essa probabilidade é proporcional à aptidão do cromossomo, sendo dada por

$$p_i = \frac{eval(v)}{F}.$$

A Figura 4.2 ilustra a roleta dividida em fatias, cada qual associada a um cromossomo.

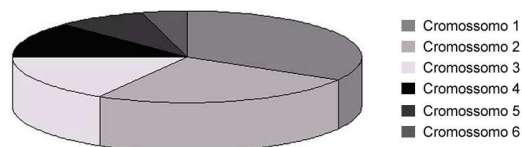


Figura 4.2: Uma roleta dividida entre 6 cromossomos.

Para facilitar o uso de um gerador de números aleatórios para a seleção dos cromossomos, calcula-se a probabilidade acumulada de cada cromossomo usando a fórmula

$$q_i = \sum_{k=1}^i (p_k), \quad i = 1, \dots, n.$$

Assim, dado um número aleatório γ entre 0 e 1, o cromossomo selecionado será aquele para o qual $q_{i-1} < \gamma \leq q_i$, supondo $q_0 = 0$.

A roleta é rodada exatamente n vezes, com o propósito de gerar uma nova população com n indivíduos. Naturalmente, pode ocorrer de um cromossomo ser selecionado mais de uma vez, o que vem de encontro com a idéia de seleção natural, já que supomos que os melhores indivíduos se reproduzem mais e os piores morrem. Outras estratégias de seleção serão vistas na Seção 5.3.

Passo 2: Cruzamento.

Um procedimento simples de cruzamento é o método de 1 ponto. Suponhamos que tenha sido definida uma probabilidade de cruzamento, p_c , que indica o número esperado de cromossomos que sofrerão esse procedimento. Neste caso, para cada cromossomo na população,

- Gera-se um número aleatório r no intervalo $[0, 1]$.
- Se $r < p_c$, seleciona-se esse cromossomo para cruzamento.

De posse do conjunto de indivíduos selecionados, estes são emparelhados (agrupados 2 a 2). Em seguida, para cada par, gera-se um número aleatório inteiro pertencente ao conjunto $\{1, \dots, n_b - 1\}$, onde n_b é o número de bits que formam um cromossomo. Esse número inteiro indica a posição de corte do par selecionado. Então, troca-se o material genético (os genes), juntando a primeira parte do primeiro cromossomo à segunda parte do segundo cromossomo, assim como a primeira parte do segundo cromossomo à segunda parte do primeiro cromossomo, como mostrado na Figura 4.3 abaixo, na qual os cromossomos P1 e P2, denominados *pais*, deram origem aos cromossomos F1 e F2, seus *filhos*.

P1: (11001 00111)	F1: (11001 11010)
P2: (00110 11010)	F2: (00110 00111)

Figura 4.3: Exemplo de cruzamento.

Passo 3: Mutação

Vejamos, finalmente, um processo extremamente simples de mutação. Para tanto, suponhamos que seja dada a probabilidade de mutação, p_m , que indica o número esperado de bits que serão selecionados para a troca de valor. Neste caso, para cada cromossomo na população atual (aquela obtida após o cruzamento) e para cada bit dentro do cromossomo,

- Gera-se um número aleatório r no intervalo $[0, 1]$.
- Se $r < p_m$, faz-se a mutação, transformando o gene de 0 para 1 ou vice-versa.

4.3 Teoria de Convergência dos Algoritmos Genéticos.

O estudo da convergência dos algoritmos genéticos é um dos maiores desafios dentro da computação evolucionária. Até o final da década de 80 do século 20, a convergência foi analisada com base nos conceitos de esquema e de blocos de construção (vide, por exemplo, Goldberg [57]), hoje já em desuso.

Em 1991, Vose e Liepins [114] mostraram que os algoritmos genéticos podem passar muitas iterações presos em soluções subótimas antes de melhorar o valor da função objetivo. Assim, o uso de um critério de parada baseado no número de iterações consecutivas sem variação da melhor solução encontrada não garante a obtenção do ótimo. Entretanto, Nix e Vose [82], modelando o comportamento dos algoritmos genéticos como Cadeias de Markov (tomando a população atual como variável de estado), mostraram que, apesar da estagnação ser uma característica dos AG, não é possível permanecer em uma solução subótima indefinidamente.

Em 1994, Rudolph [95] provou, utilizando cadeias de Markov homogêneas, que um Algoritmo Genético Canônico, como aquele apresentado na Seção 4.2.3, não converge para um ótimo global, exceto se adotada uma estratégia elitista que mantenha na população a melhor solução já encontrada.

Naturalmente, como os AG são algoritmos de busca estocásticos, não se pode garantir que uma solução ótima será encontrada em um número finito de iterações. Entretanto, em 1996, Aytug e Koehler [5] mostraram que, dado um valor $0 < \delta < 1$, é possível determinar um limite superior para o número de iterações, $t(\delta)$, de modo a garantir que a solução ótima seja visitada com probabilidade δ . Esse limite teórico, desenvolvido para o algoritmo genético canônico que apresentamos, é função da probabilidade de mutação, p_m , do tamanho da população, n , e do número de bits, γ , usado para armazenar cada cromossomo (empregando a representação binária introduzida na Seção 4.2.3).

Em [5], Aytug e Koehler provaram que

$$t_2(\delta) \leq t(\delta) \leq t_1(\delta), \quad (4.1)$$

onde

$$t_1(\delta) = \left\lceil \frac{\log(1 - \delta)}{\log(1 - \min\{(1 - p_m)^{\gamma n}, p_m^{\gamma n}\})} \right\rceil \quad (4.2)$$

e

$$t_2(\delta) = \left\lceil \max_j \left\{ \frac{\log(1 - \delta)}{\log[\rho(Q - Qe_j e_j^T)]} \right\} \right\rceil,$$

em que $\lceil x \rceil$ representa o menor valor inteiro maior ou igual a x , para $x \geq 0$; Q representa a matriz de transição da cadeia de Markov; e_j é o j -ésimo vetor canônico; e $\rho(A)$ denota o raio espectral da matrix A .

Aytug *et al.* [4] estenderam esse resultado para cromossomos com γ genes representados por números inteiros entre 0 e $K - 1$, onde K é uma potência de 2. Neste caso, o limitante $t_1(\delta)$ é dado por

$$t_1(\delta) = \left\lceil \frac{\log(1 - \delta)}{\log \left[1 - \min \left\{ (1 - p_m)^{\gamma^n}, \left(\frac{p_m}{K-1} \right)^{\gamma^n} \right\} \right]} \right\rceil. \quad (4.3)$$

Observe que esse resultado é igual a (4.2) se trabalhamos com $K = 2$.

Em 2000, Greenhalgh e Marshall [58] melhoraram o limitante de Aytug *et al.* [4], provando que é possível substituir $t_1(\delta)$ em (4.1) por

$$\tilde{t}_1(\delta) = \left\lceil \frac{\log(1 - \delta)}{n \log \left[1 - \min \left\{ (1 - p_m)^{\gamma-1} \left(\frac{p_m}{K-1} \right)^{\gamma^n}, \left(\frac{p_m}{K-1} \right)^\gamma \right\} \right]} \right\rceil. \quad (4.4)$$

Além de $\tilde{t}_1(\delta)$ ser menor que $t_1(\delta)$, este novo limitante tem como vantagem o fato de decrescer com o crescimento do tamanho da população, ao contrário que ocorre com o $t_1(\delta)$. Além disso, Greenhalgh e Marshall não exigem que K seja uma potência de 2.

Geralmente, a probabilidade de mutação p_m é pequena, de modo que o termo p_m^γ é muito pequeno. Neste caso, podemos simplificar (4.3) e (4.4) considerando que $\log(1 + x) \approx x$ para x pequeno e que $\lceil y \rceil \approx y$ para y grande. Obtemos, assim,

$$t_1(\delta) \approx - \frac{(K - 1)^{\gamma^n} \log(1 - \delta)}{p_m^{\gamma^n}} \quad (4.5)$$

e

$$\tilde{t}_1(\delta) \approx - \frac{(K - 1)^\gamma \log(1 - \delta)}{n p_m^\gamma}. \quad (4.6)$$

Supondo $K = 2$; $p_m = 0,1$; $\gamma = 8$ e $n = 100$, obtemos $t_1(0,9) \approx 2,3 \times 10^{800}$, enquanto $\tilde{t}_1(0,9) \approx 2,3 \times 10^6$, comprovando a superioridade do novo limitante.

Infelizmente, embora $\tilde{t}_1(\delta)$ seja excessivamente alto, Greenhalgh e Marshall [58] apresentaram exemplos que indicam que, para uma função objetivo geral, não é possível melhorar (4.4).

Algoritmo Genético para Roteamento com Janela de Tempo

Nos capítulos anteriores, descrevemos as principais técnicas de solução de problemas de roteamento de veículos, incluindo os algoritmos genéticos. Apresentaremos, agora, o algoritmo que adaptamos para a resolução do PRVJT, destacando, a cada tópico, alguns procedimentos interessantes encontrados na literatura.

5.1 Representação do cromossomo.

Alguns artigos da literatura propõem a representação das rotas de forma binária. Dentre eles, destacamos o de Thangiah [107], que utiliza os AG para agrupar os clientes em setores, um para cada veículo, e o texto de Benyania e Potvin [12], que emprega os algoritmos genéticos para otimizar os parâmetros utilizados na heurística de inserção de Solomon [101]. Entretanto, essa representação possui algumas desvantagens, visto que a alteração de um simples bit pode resultar em um roteiro ilegal, tornando indispensável o uso de algoritmos de reparo. A fim de tornar a manipulação do problema mais natural, os artigos mais recentes representam as rotas como vetores de números inteiros, contendo, em muitos casos, delimitadores.

Ao resolver um problema de roteamento, Barrie *et al.* [10] se baseou na representação desenvolvida por Chu e Beasley [28] para um problema de atribuição generalizado (*Generalised Assignment Problem*), que utiliza um cromossomo com n genes, cada qual associados a um cliente. Neste caso, o gene armazena o veículo com o qual o cliente será servido, ou seja, se a primeira posição contiver o número 4, o cliente 1 será servido pelo veículo correspondente ao número 4.

Apesar dessa representação evitar o uso de algoritmos de reparo, ela não torna clara a ordem em que os clientes são visitados pelos veículos. Assim, no algoritmo desenvolvido para esse trabalho, decidimos empregar a *representação por caminho*, na qual a seqüência de genes que compõem o cromossomo contém as rotas adotadas pelos veículos, ou seja, a lista ordenada dos clientes por eles visitados.

Tomemos o exemplo ilustrado na Figura 5.1. Lembrando que o depósito é representado pelo valor zero, a representação da solução apresentada na figura seria dada pelo vetor

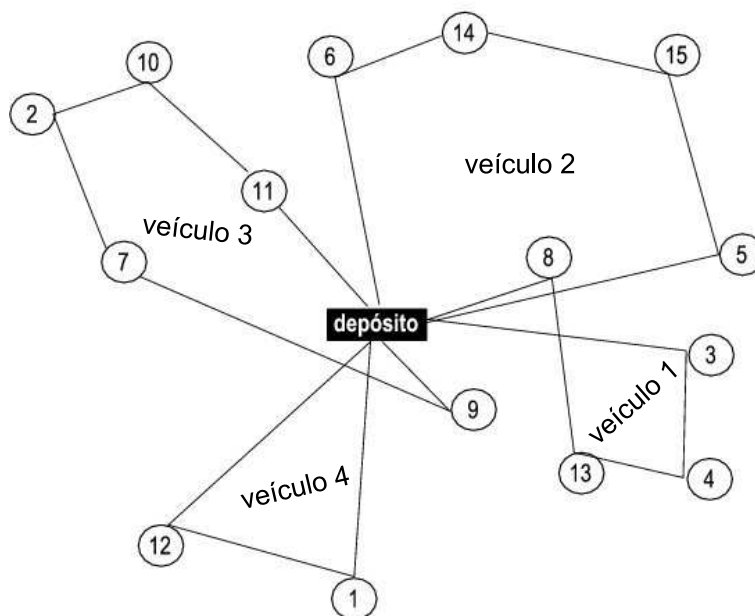
$$(3 \ 4 \ 13 \ 8 \ 6 \ 14 \ 15 \ 5 \ 9 \ 7 \ 2 \ 10 \ 11 \ 12 \ 1).$$


Figura 5.1: Uma solução para o PRVJT.

Como se observa, não usamos delimitadores para indicar o término ou início das rotas. Dessa forma, para obter tais informações, tomamos os nós na ordem fornecida pelo cromossomo e os

incluímos em uma rota até que a restrição de capacidade seja violada. Com isso, existe uma chance de não obtermos a melhor solução possível com os genes na ordem designada. No entanto, essa solução pode ser considerada satisfatória, já que a minimização do número de rotas costuma estar associada à minimização do custo total.

5.2 População Inicial.

Muitas formas de obtenção da população inicial são apresentadas na literatura, geralmente com o uso de alguma heurística simples para o problema de roteamento. As principais heurísticas utilizadas para obtenção da solução inicial são aquelas descritas no Capítulo 3. Em geral, após a aplicação de uma heurística de construção, uma ou mais técnicas de refinamento (ou a própria heurística de construção com critérios diferentes) é aplicada para obter uma porcentagem da população inicial. O restante dos cromossomos costuma ser gerado de forma aleatória, para aumentar a diversificação da população, seguindo a sugestão de Bräysy [17, 18].

Para a construção de uma solução inicial, alguns autores empregam uma técnica denominada *cluster first second method*, segundo a qual os clientes são primeiramente divididos em *clusters*, após o que são construídas as rotas. Para resolver um problema com janela de tempo, Thangiah [107, 108] contrói *clusters* através de um AG, e então vale-se da estratégia de inserção mais barata para formar as rotas.

Muitos autores usam diretamente as idéias propostas por Solomon [101] para construir uma solução. A heurística de inserção, por exemplo, é usada nos artigos de Potvin and Bengio [89] e Jung and Moon [68], nos quais os parâmetros são ajustados aleatoriamente a fim de gerar a população. Tan *et al.* [105] e Zhu [118] empregam essa heurística da mesma forma adotada nesse trabalho, a qual apresentaremos abaixo. Cordone e Calvo [32] também desenvolveram uma heurística que emprega as idéias de Solomon [101], embora definindo os custos de forma diferente. Berger *et al.* [13], por sua vez, empregam a heurística do vizinho mais próximo. Já Homberger e Gehring [63, 64, 47, 48] arquitetaram uma aproximação estocástica baseada no algoritmo de economias de Clark e Wright [29].

Neste trabalho, a população inicial é gerada seguindo os seguintes passos:

- Primeiramente, constrói-se uma solução inicial através do método *Push Forward Insertion Heuristic* (PFIH), introduzido por Solomon [101].
- Em seguida, usa-se a heurística de refinamento α -Interchange para a obtenção de uma parte da população, que é complementada com soluções obtidas de forma aleatória.

5.2.1 Construção da Solução Inicial.

Push Forward Insertion Heuristic

Antes de apresentar esta heurística de construção, é preciso introduzir alguns conceitos:

- o tempo de início de atendimento do cliente j é dado por $b_j = \max\{e_j, b_i + s_i + t_{ij}\}$;
- o custo c_{ij} é definido como $c_{ij} = \beta_1 d_{ij} + \beta_2 (b_j - b_i)$ para $\beta_1, \beta_2 \geq 0$;
- assumimos que há um número ilimitado de veículos;
- os veículos deixam o depósito no tempo e_0 .

Como as demais heurísticas de inserção, a PFIH adiciona à rota atual clientes ainda não relacionados a nenhuma rota. Para sermos mais específicos, definamos uma rota parcial factível (i_0, i_1, \dots, i_m) , na qual i_0 e i_m representam o depósito. Seja u um cliente ainda não adicionado a uma rota. Suponha, ainda que, o nó u será incluído entre os nós i_{p-1} e i_p que pertencem à rota. Observe que isso só poderá ocorrer se a rota continuar factível. Para a análise de factibilidade, entretanto, não basta verificar as restrições de capacidade e de janela de tempo para o nó que será inserido, uma vez que, a partir do nó i_p , os instantes de início de atendimento, b_{i_k} , $k = p, \dots, m$, podem ter seus valores alterados.

Com a finalidade de facilitar a verificação de factibilidade de uma rota, após a inserção de um cliente, Solomon [101] criou um procedimento denominado *push forward*, que tem como objetivo transferir aos clientes posteriores a u , o acréscimo de tempo que a inclusão deste acarreta nos próximos atendimentos. Assim, se $b_{i_p}^{new}$ é o novo instante de início de atendimento do cliente i_p , definimos o *push forward* deste cliente como

$$PF_{i_p} = b_{i_p}^{new} - b_{i_p}.$$

Para os próximos clientes da rota, esse aumento no tempo é dado por

$$PF_{i_k} = \max\{0, PF_{i_{k-1}} - w_{i_k}\}, \quad k = p + 1, \dots, m,$$

onde $w_{i_j} = b_{i_j} - b_{i_{j-1}} - s_{i_{j-1}} - t_{i_{j-1}i_j}$ é o tempo de espera para o início do atendimento do cliente i_k . Note que podemos ter $PF = 0$. Isto ocorre quando o tempo de espera é maior ou igual ao aumento ocorrido no PF anterior. Neste caso, não haverá alterações no início de atendimento dos próximos clientes. A inclusão do depósito ao final da rota tem a finalidade de garantir que o veículo retorne dentro do intervalo de tempo especificado.

Heurística de Inserção

No item anterior, mostramos uma forma eficiente encontrada por Solomon [101] para testar a factibilidade ao se adicionar um novo cliente em uma rota parcial. Agora, descreveremos os critérios utilizados para a escolha do cliente que será inserido e em qual posição isso ocorrerá na rota parcial.

Solomon [101] testou alguns critérios para a escolha do cliente que iniciará cada rota, tais como

- O nó que tem o menor instante de início da janela de tempo.
- O nó que tem o menor instante final da janela de tempo.
- O nó que se encontra mais distante do depósito.

Além disso, Solomon [101] também analisou a possibilidade de se ponderar a janela de tempo e a distância ao depósito. Como os melhores resultados foram aqueles obtidos pelas duas últimas estratégias apresentadas acima, decidimos empregá-las em nosso trabalho.

Qualquer que seja o critério adotado, suponha que o nó i_1 tenha sido escolhido para integrar a rota que, até então, estava vazia. Assim, a primeira rota parcial será (i_0, i_1, i_m) .

De posse de uma rota parcial inicial, dois custos, $c_1(i, u, j)$ e $c_2(i, u, j)$, são usados para incluir um cliente u entre dois nós sucessivos i e j da rota.

A cada cliente ainda não relacionado a uma rota, definimos seu melhor ponto de inserção na rota utilizando o custo

$$c_1(i(u), u, j(u)) = \min\{c_1(i_{p-1}, u, i_p)\}, \quad p = 1, \dots, m.$$

Em outras palavras, para cada cliente u , calculamos o custo $c_1(i_{p-1}, u, i_p)$ que esse cliente acarreta se adicionado entre os nós consecutivos i_{p-1} e i_p , $p = 1, \dots, m$. Escolhemos, então a posição que implica no menor custo.

Depois de definida a melhor posição de inserção de cada cliente, utilizamos um segundo critério para escolher, dentre todos os clientes possíveis, aquele que pertencerá à rota. Segundo Solomon [101], esse novo cliente deve ser determinado resolvendo-se o problema

$$\begin{aligned} \min \quad & c_2(i(u), u, j(u)) \\ \text{sujeito a} \quad & u \text{ ainda não tenha sido selecionado} \\ & \text{e a nova rota permaneça factível.} \end{aligned}$$

O cliente u^* selecionado é incluído entre os nós $i(u^*)$ e $j(u^*)$ da rota. Quando não for mais possível encontrar um ponto de inserção factível para algum nó u , consideramos que a rota atual está completa e uma nova rota é iniciada.

Resta-nos descrever como c_1 e c_2 são calculados. Várias fórmulas para esses custos foram sugeridas na literatura. Em geral, elas envolvem uma combinação ponderada do que se deseja priorizar, tal como a distância e o tempo de espera. Algumas destas fórmulas podem ser vistas no artigo de Solomon [101]. Neste trabalho, tomamos o conjunto de custos que obtiveram o melhor desempenho nos testes de Solomon. Esses custos são definidos por

$$c_{11}(i, u, j) = d_{iu} + d_{uj} - \mu d_{ij} \quad \text{para } \mu \geq 0 \quad (5.1)$$

$$c_{12}(i, u, j) = b_j^{\text{new}} - b_j \quad (5.2)$$

$$c_1(i, u, j) = \alpha_1 c_{11}(i, u, j) + \alpha_2 c_{12}(i, u, j); \quad \alpha_1 + \alpha_2 = 1 \quad \text{e} \quad \alpha_1, \alpha_2 \geq 0 \quad (5.3)$$

$$c_2(i, u, j) = \gamma d_{0u} - c_1(i, u, j), \quad \text{para } \gamma \geq 0 \quad (5.4)$$

Em (5.1), dá-se ênfase ao acréscimo que a adição do cliente u produzirá sobre o comprimento (ou distância) total da rota. Em (5.2), avalia-se a variação do instante de chegada no cliente j provocada pela inserção de u . A função (5.3) é a ponderação entre (5.1) e (5.2), utilizada para

a escolha do melhor ponto de inserção para cada cliente. Já (5.4) indica a economia obtida quando o cliente u é servido na mesma rota que i e j , em oposição ao custo do serviço direto de u a partir do depósito.

5.2.2 Geração da População.

Como mencionamos no início desse capítulo, parte da população é selecionada na vizinhança da solução inicial gerada pelo PFIH. A escolha dos indivíduos é feita analisando-se todos os vizinhos da solução inicial e selecionando-se os melhores cromossomos. Neste trabalho, a vizinhança é obtida através da heurística denominada λ -interchange.

λ -Interchange

Essa heurística, introduzida na Seção 3.2, foi desenvolvida por Osman [86]. Nela, a vizinhança de uma solução é criada por meio de trocas entre subconjuntos de clientes pertencentes a pares diferentes de rotas.

Seja, portanto, $S = R_1, R_2, \dots, R_k$ uma solução para o PRVJT, na qual R_i representa a rota do veículo i . A cada passo do algoritmo, selecionamos duas rotas e trocamos m_1 clientes da primeira rota por m_2 clientes da segunda rota. Podemos, assim, representar essa troca através do operador (m_1, m_2) .

O parâmetro λ indica o número máximo de clientes selecionados para a troca entre duas rotas. A fim de diminuir a complexidade do problema, usualmente λ assume o valor 1 ou 2. Vamos supor, então, que $\lambda = 2$, de modo que no máximo dois clientes de cada rota serão trocados. Assim, os possíveis operadores de troca são $(0,1)$, $(0,2)$, $(1,0)$, $(1,1)$, $(1,2)$, $(2,0)$, $(2,1)$ e $(2,2)$.

Tomemos duas rotas distintas R_p e R_q e o operador $(2,1)$ como exemplo. Neste caso, dois clientes serão selecionados na primeira rota, R_p , e apenas um cliente será retirado da segunda rota, R_q . Naturalmente, após a troca desses clientes, a solução deve continuar factível. Na Figura 5.2, encontra-se um exemplo deste procedimento. Na primeira rota, a cadeia com os nós 5 e 6 é escolhida, enquanto, na segunda rota, a cadeia removida é formada apenas pelo nó

4. Dentre as possíveis trocas de nós, a solução obtida é aquela que resulta no melhor custo total.

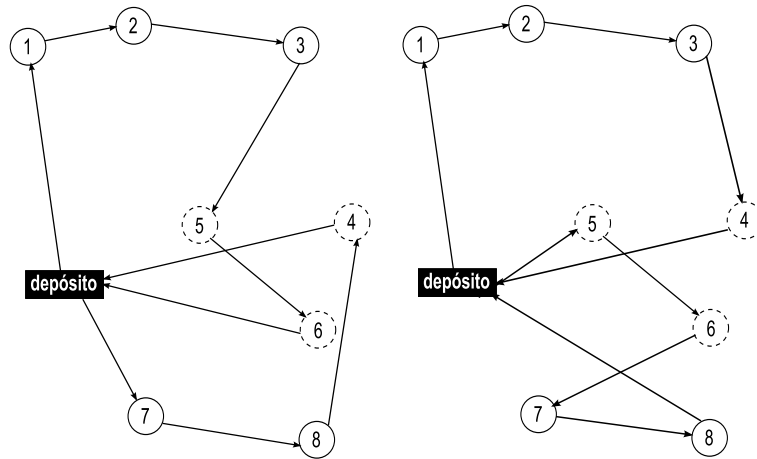


Figura 5.2: Exemplo do 2-Interchange com o operador (2,1).

Como a estratégia utilizada para vasculhar a vizinhança é a *global best*, ou seja, todos os vizinhos são analisados, devemos aplicar, a cada par de rotas, todos os operadores de troca. Dado, que o número de rotas é k , temos $k(k-1)/2$ pares de rotas possíveis, o que nos conduz a um algoritmo de complexidade $O(n^2)$.

No algoritmo desenvolvido nesse trabalho, os subconjuntos de clientes selecionados são formados apenas por nós consecutivos, que permanecem unidos na rota em que são inseridos. Além disso, a cadeia de nós retirada de uma rota é inserida, na outra rota, na posição em que o custo seja mínimo. Em nossos testes, usamos sempre $\lambda = 2$.

Depois de encontrar uma solução através do PFIH e de selecionar os melhores indivíduos vindos da vizinhança desta solução, formamos o resto da população gerando soluções aleatórias. A proporção entre boas soluções e soluções aleatórias é um parâmetro importante do algoritmo genético, uma vez que um número alto de soluções vizinhas à solução inicial pode impedir que obtenhamos uma solução distante desta região, enquanto o aumento no número de soluções aleatórias, apesar de nos conduzir a uma maior variedade na população inicial, faz com que a taxa de convergência se torne lenta. Testes realizados para a obtenção desta proporção serão apresentados no próximo capítulo.

5.3 Seleção.

A teoria da seleção natural baseia-se na idéia de que os indivíduos mais aptos passam para as gerações seguintes. Um processo muito utilizado na literatura para selecionar cromossomos pertencentes a uma população é o método da roleta, descrito na Seção 4.2.3. Entretanto, este método pode fornecer como resultado uma população excessivamente homogênea, pois os melhores indivíduos têm maior probabilidade de serem escolhidos, já que a eles são destinadas parcelas maiores da roleta. Para evitar a convergência prematura do AG decorrente dessa homogeneidade da população, Baker [8] desenvolveu uma adaptação do método na qual, em lugar de selecionar cada elemento usando um gerador aleatório, a escolha é feita supondo que os valores obtidos ao girar a roleta são equidistantes, o que reduz a chance de um mesmo indivíduo ser selecionado muitas vezes. A esse método dá-se o nome de *Amostragem Universal Estocástica*.

Um exemplo dos dois métodos baseados na roleta é dado na Figura 5.3, na qual as linhas tracejadas apontam para os indivíduos selecionados. Na primeira figura, as linhas que cortam os cromossomos selecionados assumem posições aleatórias, simulando os vários giros da roleta. Já na figura da direita, o ângulo entre as linhas é constante.

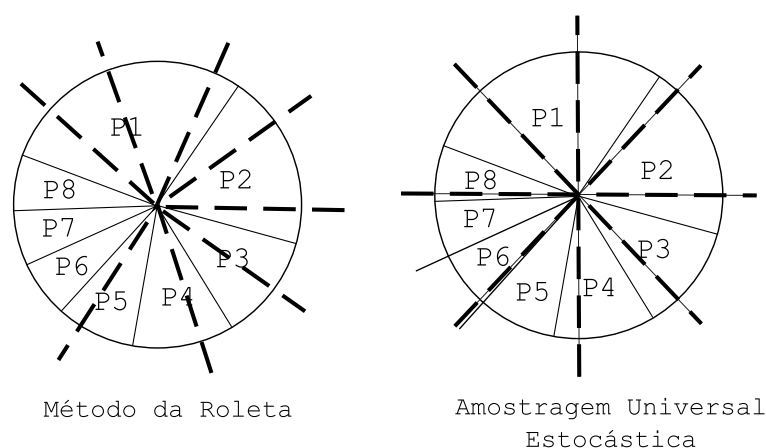


Figura 5.3: Comparação entre as versões da seleção por roleta.

Outro processo bastante utilizado para seleção é o torneio. Neste processo, faz-se k cópias da população, reordenando de forma aleatória os cromossomos de cada uma dessas cópias. Então, compara-se um conjunto de k cromossomos adjacentes, selecionando aquele com a melhor

aptidão para formar a nova população. Assim para cada cópia da população, obtém-se n/k filhos. Como temos k populações, ao final do processo, n indivíduos serão escolhidos para formar a nova população.

Na Figura 5.4, o procedimento é exemplificado para $k = 2$, o valor usado neste trabalho. No exemplo, os cromossomos i_1, \dots, i_p são selecionados da primeira cópia da população, e os cromossomos j_1, \dots, j_p são selecionados da segunda cópia, tomando $p = n/2$.

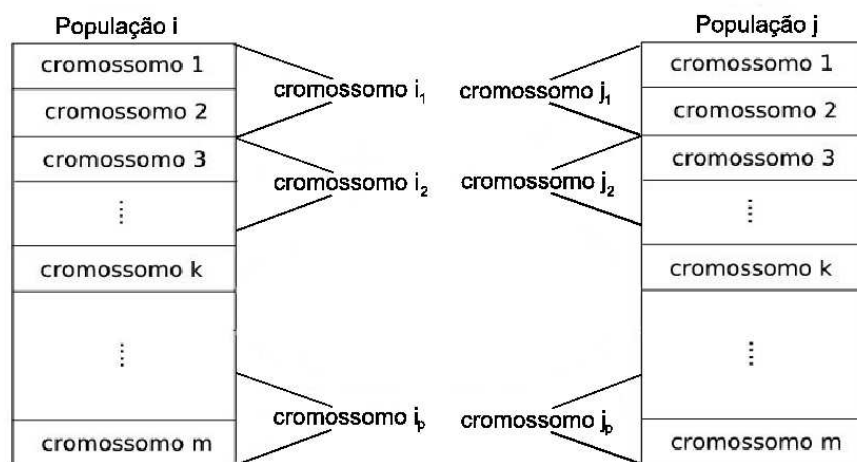


Figura 5.4: Seleção por torneio binário.

Em alguns artigos, uma estratégia elitista é empregada com o fim de garantir que bons indivíduos continuem na população. Assim, ao final do processo de seleção, um número predefinido dos piores indivíduos da nova população é substituído pelos melhores cromossomos da população antiga.

5.4 Reprodução.

A reprodução é a parte mais importante da cadeia evolutiva, pois é nela que as características são combinadas e passadas para as próximas gerações. Diversas técnicas foram propostas para a reprodução de cromossomos nos algoritmos genéticos, variando desde uma simples troca de genes até estratégias baseadas nas heurísticas de refinamento.

Duas são as operações de reprodução: o cruzamento (ou *crossover*) e a mutação.

5.4.1 Cruzamento.

Como já visto, é no cruzamento que os cromossomos passam seu material genético para seus filhos. Existem na literatura diversas formas de promover o cruzamento. Muitas delas surgiram de adaptações dos tradicionais procedimentos desenvolvidos para as representações binárias, como a proposta por Thangiah [107], que é baseada em pontos de cortes, ou seja, nas posições escolhidas para se dividir o cromossomo. Nestas estratégias, os genes compreendidos entre os pontos de corte são permutados entre os pais para gerar os novos indivíduos.

Os métodos baseados em pontos de corte podem diferenciar-se não só pela quantidade de pontos, mas também pelas diferentes maneiras de organizar as cadeias de genes a serem inseridas nos novos cromossomos. Existem procedimentos que tentam preservar a seqüência de genes provenientes dos pais, assim como há procedimentos que preservam a posição dos genes vinda dos pais. Um exemplo de cruzamento que busca manter a ordem e posição dos genes é o PMX. Este método foi selecionado, juntamente com o *cruzamento heurístico* e o *cruzamento por fusão*, para compor o algoritmo genético que desenvolvemos neste trabalho. Fazemos abaixo uma descrição dos principais tipos de cruzamento, incluindo aqueles que empregamos em nosso programa.

A probabilidade para a escolha dos cromossomos que sofrerão o cruzamento é definida *a priori*. Segundo a literatura, esse valor deve ser alto e maior que a probabilidade de mutação.

PMX (Partially Mapped Crossover).

Proposto por Goldberg e Lingle [56], este método utiliza 2 pontos de corte. Para cada par de pais, definimos os pontos de corte aleatoriamente e permutamos o material genético contido entre eles para gerar os filhos.

Entretanto, deve-se observar que, durante a troca de material genético, é possível que os cromossomos passem a ter genes repetidos. Neste caso, substituímos cada valor repetido (fora da região interna ao corte) pelo gene pertencente ao mesmo locus do corte no outro cromossomo.

Para melhor entender o procedimento, considere o casal de cromossomos e os cortes dados por

P1: (1 2 3 | 4 5 6 | 7 8 9);

$$P2: (5\ 3\ 1 \mid 2\ 6\ 9 \mid 7\ 4\ 8).$$

Neste caso, trocando os genes (4 5 6) de P1 pelos genes (2 6 9) de P2, obtemos os seguintes filhos:

$$F1': (1\ 2\ 3 \mid 2\ 6\ 9 \mid 7\ 8\ 9);$$

$$F2': (5\ 3\ 1 \mid 4\ 5\ 6 \mid 7\ 4\ 8).$$

Nota-se a repetição dos genes 2 e 9 no primeiro filho e 4 e 5 no segundo filho, o que torna infactível a solução. Para contornar esse problema, substituímos cada gene repetido pelo valor na mesma posição dentro do corte, mas no outro filho. Assim, substituímos o 2 pelo 4, o 6 pelo 5 e o 9 pelo 6, obtendo:

$$F1: (1\ 4\ 3 \mid 2\ 6\ 9 \mid 7\ 8\ 5);$$

$$F2: (9\ 3\ 1 \mid 4\ 5\ 6 \mid 7\ 2\ 8).$$

Deve-se observar que os genes compreendidos entre os cortes permanecem intactos, sendo as trocas feitas nos genes externos aos cortes. Além disso, nota-se, no exemplo acima, que a troca do 9 pelo 6 em F1' ainda não produziria uma solução factível, motivo pelo qual foi preciso aplicar também a troca entre o 6 e o 5 para que a solução F1 fosse finalmente obtida. Problema similar ocorreu no cromossomo F2', de modo que primeiro gene de F2' foi convertido de 5 para 6 e, em seguida, de 6 para 9.

OX (Order Crossover) e OBX.

O Order Crossover (OX), proposto por Davis [35], conservar apenas a ordem dos genes. Este cruzamento foi utilizado para resolver o PRVJT por Zhu [119] (que também implementou o PMX) e por Bouthillier e Crainic [16]. O *Order Based Crossover* (OBX), uma modificação do OX, foi aplicado ao PRVJT por Homberger e Gehring [63], a partir de uma adaptação do modelo de Goldberg [57].

Apresentamos, a seguir, o procedimento adotado no OX, juntamente com sua aplicação ao par de pais P1: (1 2 3 4 5 6 7 8 9) e P2: (4 5 2 1 8 7 6 9 3).

1. Escolhe-se aleatoriamente 2 pontos de corte. Em nosso exemplo, serão usadas as posições 3 e 7.
2. Começa-se a construir os dois filhos mantendo os genes compreendidos entre os cortes:
F1: (X X X 4 5 6 7 X X) e F2: (X X X 1 8 7 6 X X).
3. Retira-se do pai 2 os genes já existentes no filho 1. Em nosso caso, devemos retirar os genes 4 5 6 e 7, de modo que obtemos P2': (X X 2 1 8 X X 9 3).
4. Toma-se os genes remanescentes de P2' seguindo a ordem iniciada pelo primeiro gene ainda não adicionado após o segundo corte. Procedendo assim no nosso exemplo, tomamos os genes 9, 3, 2, 1 e 8 de P2', nessa ordem.
5. Insere-se os genes obtidos em F1, na ordem dada acima, partido do primeiro locus vazio após o segundo corte. Obtemos assim, F1: (2 1 8 4 5 6 7 9 3);
6. Aplica-se procedimento similar para a obtenção de F2. No nosso exemplo, F2: (3 4 5 1 8 7 6 9 2).

As diferenças entre o OBX e o OX concentram-se em dois pontos. Primeiramente, o OBX utiliza múltiplos pontos de corte, em lugar de apenas 2. Além disso, no OBX, os genes retirados do cromossomo pai no Passo 4 e inseridos no filho no Passo 5 acima são ordenados a partir do início do cromossomo, em lugar do primeiro locus após o último corte.

Tomando os mesmos pais e seguindo os mesmos passos do exemplo acima, o procedimento OBX poderia ser resumido nos seguintes passos:

1. Pontos de corte: 1,2,4,6 e 8.
2. Formação inicial dos filhos: F1: (X 2 X X 5 6 X X 9) e F2: (X 5 X X 8 7 X X 3).
3. Pais sem os genes já transferidos aos filhos: P2': (4 X X 1 8 7 X X 3) e P1': (1 2 X 4 X 6 X X 9).
4. Genes a serem inseridos: em F1: 4, 1, 8, 7 e 3; em F2: 1, 2, 4, 6 e 9.
5. Filhos obtidos: F1: (4 2 1 8 5 6 7 3 9) e F2: (1 5 2 4 8 7 6 9 3).

Cruzamento Heurístico (*Heuristic Crossover*).

Este método difere um pouco dos anteriores, pois não se baseia em uma simples troca de genes, além de gerar apenas um filho como resultado da combinação de dois pais. O método é iniciado fazendo-se um corte aleatório para cada par, como no exemplo abaixo.

P1: (1 2 3 4 | 5 6 7 8 9);

P2: (5 3 1 2 | 6 9 7 4 8).

Em seguida escolhe-se o primeiro gene após o corte de um dos pais (vamos supor que o primeiro gene de P1 seja o escolhido) para pertencer ao filho (no exemplo, esse gene é o 5). Então, localiza-se no segundo pai o gene igual ao escolhido, substituindo-o pelo primeiro gene após o corte neste pai (que, em nosso exemplo, é o 6). Para os cromossomos mostrados acima, temos

P1': (1 2 3 4 | 5 6 7 8 9);

P2': (**6** 3 1 2 | **5** 9 7 4 8).

Agora os dois pais têm o mesmo gene na posição seguinte ao corte. Passamos, então, ao gene seguinte de cada cromossomo. No nosso exemplo, esses genes são o 6 em P1' e o 9 em P2'. Dentre esses genes, o cromossomo filho herdará aquele que possui a menor distância ao gene escolhido no passo anterior. Seguindo o exemplo, analisamos a distância entre o nó 5 e os nós 6 e 9. Supondo que $d_{56} \leq d_{59}$, escolhemos o gene 6, que permanece em sua posição. Para substituir o gene 9 pelo 6 em P2', usamos o processo descrito no parágrafo acima, obtendo

P1'': (1 2 3 4 | 5 6 7 8 9);

P2'': (**9** 3 1 2 | 5 **6** 7 4 8).

Esse procedimento é repetido até que todos os genes tenham sido selecionados. Naturalmente, consideramos que os genes formam uma lista circular, de modo que, depois de visitar o último locus do cromossomo, continuamos a partir do primeiro gene. Ao final do cruzamento, os dois cromossomos são iguais. A este procedimento denominamos Cruzamento Heurístico 1 (CH1).

Em uma variante deste método, chamada Cruzamento Heurístico 2 (CH2), ao invés de promover a troca de genes no pai que não teve seu gene escolhido, o gene selecionado em um pai é eliminado do outro, independentemente de sua posição. Vamos ilustrar esse método usando os mesmos pais do exemplo acima.

1. Seguindo o mesmo critério do CH1, escolhemos o gene 5 de P1. Entretanto, em lugar de substituir o gene 6 pelo 5 em P2, eliminamos o gene 5 do segundo pai:

$$P1': (1\ 2\ 3\ 4\ | \mathbf{5}\ 6\ 7\ 8\ 9);$$

$$P2': (x\ 3\ 1\ 2\ | 6\ 9\ 7\ 4\ 8).$$

2. No pai que teve o gene escolhido no passo anterior, tomamos o próximo gene (no exemplo, tomamos o gene 6 de P1'). No outro pai, tomamos o mesmo gene utilizado no passo anterior (no exemplo, o gene 6 de P2'). Dessa forma, devemos selecionar um gene entre o 6 de P1' e o 6 de P2'. Optando pelo gene do primeiro pai e executando o procedimento de exclusão do gene 6 de P2', obtemos

$$P1'': (1\ 2\ 3\ 4\ | \mathbf{5}\ \mathbf{6}\ 7\ 8\ 9);$$

$$P2'': (x\ 3\ 1\ 2\ | x\ 9\ 7\ 4\ 8).$$

3. Repetimos o procedimento até que todos os genes tenham sido utilizados.

Cruzamento por Fusão (*Merge Crossover*).

Este método segue os mesmos princípios do Cruzamento Heurístico. Entretanto, em lugar de compararmos a distância, avaliamos o tempo limite em que um cliente pode receber a visita de um veículo. Assim, o gene escolhido é aquele que possui o menor instante final da janela de tempo. O procedimento em que ocorre uma troca de genes similar à usada no CH1 é chamado de *Cruzamento por Fusão 1* (CF1) e aquele no qual o gene é eliminado, como no CH2, denominamos *Cruzamento por Fusão 2* (CF2).

Observa-se que os métodos CH1, CH2, CF1 e CF2 produzem apenas um filho por casal. Assim, para formar a nova população, utiliza-se a combinação de dois desses métodos. Como

exemplo, podemos utilizar CH1 e CF1 ou CH2 e CF1 para formar um par de filhos a partir de um par de pais.

Outros tipos de cruzamento.

Grenfenstette [59] desenvolveu outra classe de operadores heurísticos para a execução do cruzamento, dando ênfase às arestas e não aos nós. Seu método é composto pelos seguintes passos:

1. Escolhe-se um cliente c aleatoriamente.
2. Seleciona-se as 4 arestas que incidem no cliente c , cada par vindo de um pai.
3. Através de um critério predefinido, escolhe-se um dos clientes pertencentes às arestas definidas no Passo 2.
4. Toma-se esse novo cliente como c e retorna-se ao Passo 2.
5. Termina-se o processo quando todos os clientes tiverem sido rotacionados.

Baseados nesse esquema, Whitley, Starweather e Fuquay [117] desenvolveram o *Edge Recombination Crossover* (ER), utilizado por Bouthillier e Crainic [16] para resolver um PRVJT. A idéia central do ER é que um filho deve ser construído exclusivamente de arestas vindas dos pais. Neste caso, o cliente a ser inserido é aquele que possui o menor número de nós a ele conectados, ou seja, o menor número de arestas nele incidentes. No caso de ocorrer empate, a escolha é feita aleatoriamente.

Uma modificação desse procedimento foi utilizada por Ochi *et al.* [83], que define o primeiro cliente a ser inserido como aquele que está mais próximo do depósito. Já Hwang [65] utilizou a ordem numérica como critério de desempate dos clientes com número igual de nós incidentes. Assim, por exemplo, em caso de empate entre os clientes 2 e 5, o cliente 2 é escolhido.

Existem ainda métodos em que a construção do filho se baseia em rotas vindas dos pais. Neste caso, é necessário empregar algoritmos de reparo para tornar a solução factível. Um exemplo pode ser encontrado em Potvin e Bengio [89], que desenvolveram um procedimento que busca construir filhos com distância melhor. Esse procedimento envolve a remoção de uma aresta escolhida aleatoriamente de cada pai. Em seguida, novas arestas são criadas para

reconectar os pais, ao que se segue a aplicação de um algoritmo de reparo para remover a infactibilidade.

Idéias vindas de heurísticas de refinamento também são usadas na literatura. Exemplos são encontrados em Potvin e Bengio [89] e em Berger *et al.* [14, 13].

5.4.2 Mutaç o.

O prop sito principal da muta o   promover um aumento da diversidade dos cromossomos da popula o. Existem diversos tipos de operadores de muta o. Os m todos mais tradicionais se baseiam na id ia cl sica, vinda da representa o bin ria, de altera o do valor de genes escolhidos aleatoriamente.

Procedimentos particularmente adaptados para cromossomos compostos por genes inteiros tamb m s o encontrados na literatura. Thangiah [107], por exemplo, ap s escolher um gene, o substitui por outro gene pertencente ao cromossomo. Esse procedimento pode ser expandido para trocas de cadeias de genes de tamanhos variados. A aplica o desse tipo de procedimento ao PRVJT   encontrada em Tan *et al.*; [105], artigo no qual nos baseamos para implementar nossa vers o da muta o, conforme explicado mais adiante nesta se o.

Louis *et al.* [78] desenvolveram um algoritmo eficiente para problemas nos quais os n s se aglomeram em algumas regi es. Para tanto, utilizam uma muta o com dois tipos de troca de genes. No primeiro, somente ocorre a mudan a se os genes escolhidos pertencerem  s k primeiras posi es ou se ambos se situarem nas $(n - k)$  ltimas posi es do cromossomo. Na segunda,   escolhido um cliente pertencente  s k primeiras posi es e o outro  s $(n - k)$ posi es finais. Entretanto, para que a troca seja aceita, ambos os genes devem pertencer ao mesmo *cluster*, conjunto de clientes existente em uma regi o.

Em muitos artigos, a muta o   usada para melhorar as rotas. Nestes casos, ela se baseia em m todos heur sticos como o 2-opt e or-opt, adaptados para que, ao final do processo, haja um n mero menor de ve culos na solu o. Para mais detalhes, deve-se consultar Br ysy e Gendreau [25].

O procedimento de muta o usado neste trabalho se baseia nos m todos de Tan *et al.* [105], com uma cadeia composta por apenas um gene. Primeiramente, escolhemos aleatoriamente 2

genes e, então, fazemos a permuta entre eles, como ilustra a Figura 5.5.



Figura 5.5: A mutação.

Como já foi visto, a escolha da probabilidade de mutação, P_{mut} , é muito importante, já que valores altos nos conduzem a uma convergência lenta, enquanto valores baixos produzem uma convergência prematura a mínimos locais. Tan *et al.* [105] desenvolveram um esquema para o cálculo dessa probabilidade adaptando P_{mut} ao desvio padrão da população. Para tanto, eles definem

$$S = \sqrt{\frac{\sum_{i=0}^{n-1} (d_i - \bar{d})^2}{n-1}},$$

onde n é o tamanho da população, d_i é a distância total associada à solução i e \bar{d} é a média das distâncias da população. Se $S \geq 5$, usa-se $P_{mut} = 0,06$. Caso contrário, $P_{mut} = 0,06 + 0,18(5 - S)$. Dessa forma, há sempre a garantia de que a probabilidade seja maior ou igual a 0,06.

5.5 Procedimentos de Pós-Otimização.

Como vimos, os algoritmos genéticos são uma poderosa ferramenta para resolver o PRVJT. Entretanto, trabalhos recentes nesta área vêm mostrando a necessidade da combinação de técnicas para que se obtenha os melhores resultados. Isso se reflete no grande número de métodos híbridos surgidos nos últimos anos.

Dois procedimentos foram adotados nesse trabalho visando melhorar a qualidade da população em cada geração: o *Hill Climbing* e a Recuperação.

5.5.1 *Hill Climbing*.

O *Hill Climbing* é inspirado no método da máxima descida (ou da máxima subida, se considerarmos uma função objetivo de maximização). Trata-se de um procedimento de melhora iterativa

no qual, a cada passo, uma busca local é realizada na vizinhança da solução selecionada no passo anterior.

Para o PRVJT, construímos a vizinhança de uma solução usando o 1-Interchange, ou seja, trocando no máximo 1 cliente entre cada par de rotas. A estratégia utilizada para a aceitação de um vizinho é denominada *first best*. Segundo essa estratégia, o primeiro vizinho com solução melhor que o cromossomo corrente é aceito. Os passos do algoritmo são dados a seguir.

1. Buscar uma solução s' na vizinhança de s .
2. Se s' é melhor que s ,
 - 2.1. $s \leftarrow s'$.
 - 2.2. Voltar ao Passo 1.
3. Caso contrário, parar, pois já estamos em um mínimo local.

O principal critério usado para avaliar uma solução é a distância total. Entretanto, realizamos testes combinando, de forma ponderada, a distância total e o número de veículos.

Os experimentos mostram que esse método é bem poderoso. Contudo, ele exige um grande esforço computacional. Dessa forma, é preciso limitar a porcentagem dos cromossomos da população que são submetidos ao *Hill Climbing*. Detalhes serão fornecidos no próximo capítulo.

5.5.2 Recuperação (*Recovery*).

A troca de genes que ocorre na reprodução pode fazer com que uma boa solução seja perdida antes mesmo de ter sua vizinhança explorada. Com isso, deixa-se de buscar indivíduos em regiões promissoras.

A Recuperação (tradução do termo inglês *Recovery*), tem a finalidade de impedir que bons pais saiam da população. Ao final de cada iteração, uma porcentagem dos piores filhos gerados é substituída pelos melhores indivíduos da população inicial, os pais. Esta porcentagem deve ser pequena e menor que a da mutação, pois isso evita uma convergência prematura para um mínimo local.

5.6 O Algoritmo.

O algoritmo que implementamos neste trabalho engloba os procedimentos descritos acima, podendo ser descrito pela seguinte seqüência de passos:

1. Gerar a população inicial:
 - 1.1. Criar a solução inicial s utilizando o PFIH.
 - 1.2. Construir a vizinhança de s através do 2-Interchange.
 - 1.3. Selecionar os melhores vizinhos de s .
 - 1.4. Gerar aleatoriamente os cromossomos restantes.
2. Repetir:
 - 2.1. Seleção.
 - 2.2. Cruzamento.
 - 2.3. Mutação.
 - 2.4. Hill Climbing.
 - 2.5. Recuperação.
3. Até que o critério de parada seja satisfeito.
4. Aplicar o Hill climbing sobre o melhor indivíduo da população.

O critério de parada utilizado foi o número de iterações. Entretanto, testamos também como critério a estagnação da melhor solução por um determinado número de iterações. Os detalhes serão fornecidos no próximo capítulo.

Experimentos Computacionais

Neste capítulo, apresentamos os testes computacionais realizados com o algoritmo proposto no Capítulo 5. Esse algoritmo foi implementado em duas linguagens diferentes. Uma primeira versão, desenvolvida no ambiente Matlab, permitiu-nos analisar a influência sobre o desempenho do algoritmo genético de parâmetros tais como as probabilidades de cruzamento e de aplicação do *hill climbing*. Essa versão também foi utilizada para avaliar as diversas estratégias de *cruzamento*, como veremos na próxima seção.

Após esta análise inicial do desempenho do algoritmo em Matlab, elaboramos uma versão em c++, com o propósito de avaliar sua eficácia quando comparado a outros algoritmos apresentados na literatura.

Os testes foram feitos em um computador com processador Intel Pentium D, de 2,4 GHz, com 2 Gb de memória RAM. Os programas foram executados no sistema operacional Fedora Linux. O compilador c++ empregado foi o gnu g++.

6.1 Problemas utilizados nos testes.

Solomon [102] criou um conjunto de problemas para permitir que o comportamento de seus algoritmos de roteamento fosse avaliado. Estes problemas vêm servindo como base de comparação entre os vários algoritmos apresentados na literatura especializada, motivo pelo qual

decidimos utilizá-los, neste trabalho, para analisar o desempenho do algoritmo proposto.

O conjunto é constituído por 56 instâncias, cada uma das quais contendo 100 clientes e um depósito central. Os problemas são separados de acordo com a localização dos clientes, formando-se assim 3 grupos. No primeiro, os clientes são distribuídos de forma aleatória. No segundo, eles são agrupados em conjuntos ou *clusters*. No último, há uma mistura das duas situações anteriores.

Esses grupos, representados pelas siglas R, C e RC respectivamente, são subdivididos, ainda, em duas classes cada. Na Classe 1, as janelas de tempo, mais curtas, têm duração que varia entre 25% e 50% do intervalo entre a abertura e o fechamento do depósito. Na Classe 2, as janelas de tempo são mais longas, variando de 75% a 100% do período de funcionamento do depósito. Assim, tem-se 6 grupos de problemas representados, respectivamente, por R1, C1, RC1, R2, C2 e RC2, contendo, cada um, de 8 a 12 problemas.

6.2 Perfis de desempenho.

Além das tabelas usuais, apresentaremos os resultados dos testes utilizando um gráfico desenvolvido por Dolan e Moré [39], denominado *perfil de desempenho*.

Esse gráfico tem por finalidade facilitar a comparação de um conjunto de algoritmos, S , elaborados para resolver um conjunto de problemas, P . Essa comparação é feita com base em uma medida escolhida para a análise. Em nossos testes, essa medida é, dependendo do caso, o tempo computacional consumido, o valor da função objetivo do problema ou o número de veículos utilizados.

Para explicar como o perfil é gerado, suponhamos que a medida escolhida seja a distância total obtida pelo algoritmo, ou seja, a soma das distâncias percorridas pelos veículos para atender todos os clientes. Definamos, então, $t_{p,s}$ como a distância total encontrada pelo algoritmo s ao resolver o problema p . A *razão de desempenho* do algoritmo s com relação ao problema p , representada por $r_{p,s}$, é dada pela razão entre a distância total obtida por s e a menor distância obtida pelos algoritmos utilizados para resolver o problema p , ou seja,

$$r_{p,s} = \frac{t_{p,s}}{\min \{t_{p,s} : s \in S\}}.$$

O desempenho global do algoritmo s em relação ao conjunto P , que contém n_p problemas, é dado pela função

$$P(t) = \frac{1}{n_p} \text{card}(\{p \in P : r_{p,s} \leq t\}),$$

onde $\text{card}(C)$ é a cardinalidade do conjunto C . A função $P(t)$ indica a porcentagem do número de problemas que é resolvida por s dentro de um fator t do valor da menor distância encontrada pelo conjunto de algoritmos. Assim, se $P(1,05) = 0,9$, então, para 90% dos problemas, a solução obtida pelo algoritmo s não superou em mais de 5% a melhor solução obtida pelos algoritmos utilizados no teste. Podemos notar, também, que o valor de $P(1)$ indica o percentual de problemas para os quais o algoritmo s obteve a melhor solução.

Fazendo t variar, obtemos o gráfico de perfil de desempenho para um algoritmo. Agrupando as curvas obtidas para todos os algoritmos avaliados, é possível compará-los facilmente. Para tanto, basta observar que um algoritmo será tanto melhor quanto mais alta for a curva de seu perfil de desempenho.

6.3 Testes executados no Matlab.

Uma primeira versão completa do algoritmo foi implementada usando a linguagem Matlab. Essa implementação foi usada, principalmente, para

- A escolha da estratégia de cruzamento, entre o PMX, o cruzamento heurístico do tipo 1 ou 2 e o cruzamento por fusão do tipo 1 ou 2.
- A determinação da fração da população inicial composta por indivíduos pertencentes à vizinhança da solução inicial encontrada pelo PFIH.
- A escolha do cliente que iniciará cada rota na construção da solução inicial pelo algoritmo PFIH, entre o cliente mais distante possível e o cliente com o menor instante final da janela de tempo.
- O uso do *hill climbing*.

A escolha do valor ótimo para esses parâmetros do algoritmo é discutida separadamente nas subseções abaixo. Sempre que um dos parâmetros é analisado, os demais permanecem

fixos. Desta forma, caso não haja qualquer menção em contrário, os resultados apresentados abaixo foram obtidos utilizando o cruzamento do tipo PMX, iniciando cada rota do algoritmo PFIH pelo cliente mais distante, definindo 50% da população inicial com base na vizinhança da solução obtida pelo PFIH e aplicando o *hill climbing* com probabilidade de 50%. Em nossos testes, a combinação desses valores forneceu os melhores resultados para o algoritmo genético.

Outros parâmetros do algoritmo foram definidos de acordo com as sugestões de Tan *et al.* [105], que são seguidas por inúmeros autores. Assim, o tamanho da população foi fixado em 100 indivíduos e a probabilidade de cruzamento foi definida como 0,77.

Em virtude do elevado tempo de processamento do algoritmo no Matlab, provocado pelo uso do *hill climbing*, o número de iterações (ou gerações) foi fixado em 30.

6.3.1 Tipo de cruzamento.

Em nosso primeiro conjunto de testes, confrontamos os tipos de cruzamento implementados. Como o cruzamento heurístico (CH) e o cruzamento por fusão (CF) produzem apenas um descendente por dupla de cromossomos pais, foi preciso combinar dois desses procedimentos para que a população da geração seguinte tivesse o número correto de indivíduos. Assim, além do cruzamento PMX, analisamos seis combinações dos procedimentos CH1, CH2, CF1 e CF2. Uma análise da qualidade da solução obtida pelo algoritmo para cada tipo de cruzamento é fornecida pelo gráfico de perfil de desempenho da Figura 6.1.

Um resumo dos resultados obtidos pode ser encontrado na Tabela 6.1. Nesta, bem como nas demais tabelas apresentadas ao longo do capítulo, o termo *acumulado* indica a soma das distâncias totais obtidas para todas os problemas pertencentes ao respectivo grupo. Já o termo *média* representa o valor acumulado dividido pelo número de instâncias pertencentes ao grupo. O valor acumulado total e a média total dizem respeito às 56 instâncias de Solomon [102]. Os número em negrito correspondem aos melhores resultados obtidos.

A Tabela 6.1 e a Figura 6.1 indicam, claramente, que o método PMX teve um desempenho superior em todas as classes de problemas. Nota-se, inclusive, que as soluções obtidas por este tipo de cruzamento não superaram sequer em 1% a melhor solução obtida para cada problema.

Pelo gráfico da Figura 6.1, também concluímos que os perfis de desempenho das diver-

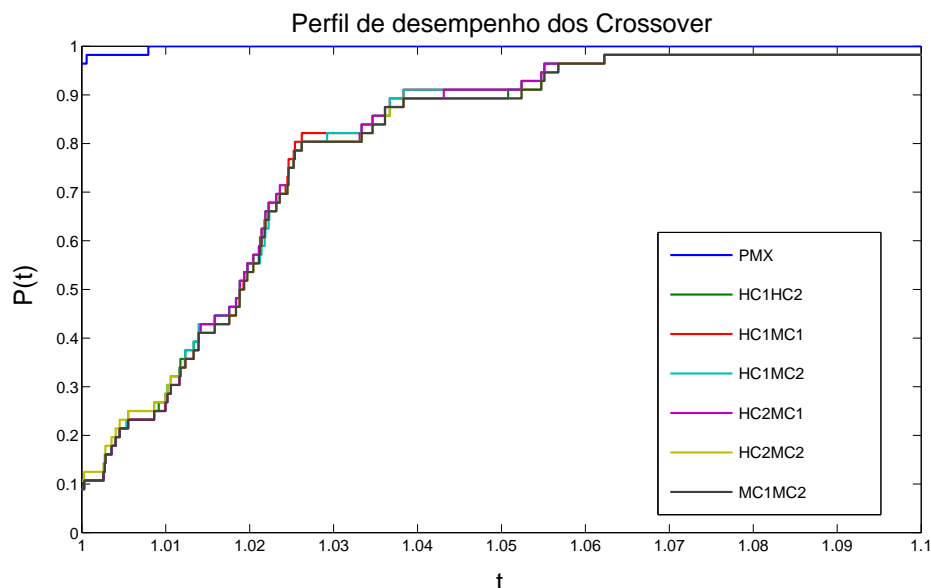


Figura 6.1: Qualidade da solução obtida pelo algoritmo para os diversos tipos de cruzamento.

As combinações do cruzamento heurístico com o cruzamento por fusão estão muito próximas, indicando desempenhos similares. Este fato também é evidenciado pelos valores médios e acumulados para os 56 problemas analisados, exibidos na Tabela 6.1, que apresentam pequena variação percentual.

6.3.2 Vizinhaça da solução inicial oriunda do método PFIH.

Em uma segunda bateria de testes, avaliamos a influência do número de vizinhos vindos da solução inicial gerada pelo método PFIH na constituição da população inicial.

Começamos construindo a população inicial apenas com cromossomos gerados aleatoriamente. Em seguida, testamos o que ocorria quando 25, 50, 75 e 100% da população inicial era composta por cromossomos vizinhos à solução fornecida pelo algoritmo PFIH. Para cada uma dessas porcentagens, resolvemos dez vezes todos os problemas proposto por Solomon, utilizando, em cada caso, uma semente diferente para nosso gerador de números aleatórios. Um resumo dos resultados obtidos está exposto na Tabela 6.2. Os perfis de desempenho são dados na Figura 6.2.

Analisando a Tabela 6.2 e o gráfico da Figura 6.2, constatamos que os resultados produzidos por populações formadas exclusivamente por indivíduos gerados aleatoriamente são indiscuti-

Classe de problemas		PMX	CH1 + CH2	CH1 + CF1	CH1 + CF2	CH2 + CF1	CH2 + CF2	CF1 + CF2
C1	Acumulado	8376,8	8531,1	8544,7	8538,5	8545,7	8545,7	8545,7
	Média	930,86	947,89	949,41	948,73	949,52	949,52	949,52
C2	Acumulado	8326,5	8478,4	8444,8	8449,9	8464,3	8478,4	8478,4
	Média	925,17	942,04	938,31	938,88	940,47	942,04	942,04
R1	Acumulado	16297,7	16658,3	16658,3	16650,6	16641,2	16644,1	16657,4
	Média	1810,86	1850,92	1850,92	1850,07	1849,02	1849,34	1850,82
R2	Acumulado	12725,5	13155,1	13138,7	13138,7	13138,7	13138,7	13138,7
	Média	1413,95	1461,67	1459,85	1459,85	1459,85	1459,85	1459,85
RC1	Acumulado	12286,1	12458,4	12458,4	12460,7	12458,4	12458,4	12458,4
	Média	1365,12	1384,26	1384,26	1384,52	1384,26	1384,26	1384,26
RC2	Acumulado	10751,6	10949,6	10949,6	10949,6	10949,6	10949,6	10949,6
	Média	1194,62	1216,62	1216,62	1216,62	1216,62	1216,62	1216,62
Acumulado Total		68764,2	70230,8	70194,4	70187,9	70197,8	70214,9	70228,2
Média Total		1227,93	1254,12	1253,47	1253,36	1253,53	1253,84	1254,07

Tabela 6.1: Comparação entre os diversos tipos de cruzamento

velmente inferiores. Por outro lado, não se nota uma diferença expressiva entre os resultados obtidos para os percentuais de vizinhos entre 50% e 100% da solução ótima do algoritmo PFIH.

De fato, não há um percentual de vizinhos que vença em todas as classes de problemas. A Tabela 6.2 sugere uma leve superioridade dos resultados gerados pelas populações compostas por 100% de vizinhos da solução do PFIH, pois esse percentual forneceu as melhores médias em três das seis classes de problemas, além de ter atingido as menores distâncias acumuladas totais. Entretanto os perfis de desempenho da Figura 6.2 revelam que as soluções obtidas usando-se 50% da vizinhança da solução fornecida pelo PFIH nunca foram mais que 4% mais caras que as melhores soluções obtidas, enquanto esse valor sobe para 5% quando se usa 75% da vizinhança do PFIH e para 7% quando 100% da solução inicial é composta por vizinhos da solução fornecida pelo PFIH.

Classe de problemas		Percentual de vizinhos oriundos de PFIH				
		0	25	50	75	100
C1	Acumulado	8698,2	8471,9	8472,3	8498,7	8489,3
	Média	966,47	941,32	941,36	944,30	943,26
C2	Acumulado	8618,9	8344,3	8353,6	8468,6	8332,3
	Média	1077,36	1043,04	1044,20	1058,57	1041,54
R1	Acumulado	16871,3	16369,4	16307,5	16373,4	16333,7
	Média	1405,94	1364,11	1358,96	1364,45	1361,14
R2	Acumulado	13574,9	12802,5	12793,7	12786,0	12772,4
	Média	1234,08	1163,86	1163,06	1162,36	1161,12
RC1	Acumulado	12596,6	12328,3	12291,2	12276,3	12296,3
	Média	1574,57	1541,04	1536,40	1534,54	1537,04
RC2	Acumulado	11132,2	10912,0	10814,9	10787,9	10746,2
	Média	1391,52	1364,00	1351,87	1348,49	1343,27
Acum. Total		71492,0	69228,4	69033,2	69190,8	68970,2
Média Total		1276,64	1236,22	1232,74	1235,55	1231,61

Tabela 6.2: Comparação da porcentagem de vizinhos vindos da solução inicial fornecida pelo algoritmo PFIH na constituição da população inicial

Assim sendo, para assegurar uma maior diversidade da população, decidimos adotar uma população inicial que tem 50% de seus indivíduos gerados aleatoriamente e 50% oriundos da vizinhança da solução produzida pelo PFIH.

6.3.3 Cliente inicial das rotas geradas pelo algoritmo PFIH.

Como foi dito anteriormente, julgamos que seria pertinente discutir qual critério o algoritmo PFIH deveria usar para determinar o cliente inicial de cada rota. Duas opções foram consideradas a partir da análise da literatura sobre o assunto: a escolha do cliente mais distante do depósito ou a escolha do cliente cujo instante final da janela de tempo era menor. Os resultados comparativos dessas alternativas são dados na Tabela 6.3 e na Figura 6.3.

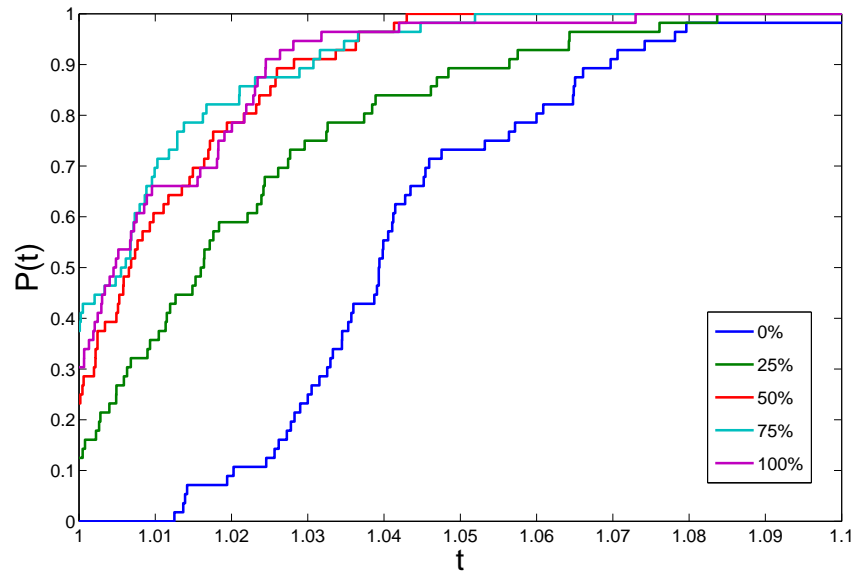


Figura 6.2: Influência do percentual de vizinhos da solução fornecida pelo algoritmo PFIH sobre a solução obtida pelo algoritmo genético.

A Tabela 6.3 aponta claramente a superioridade da escolha do cliente mais distante do depósito como ponto de partida para as rotas geradas pelo algoritmo PFIH. Esse critério apresentou desempenho superior em todos os problemas do Solomon [102]. O gráfico da Figura 6.3 também indica que esta opção forneceu a melhor solução em cerca de 90% dos problemas. Além disso, sua curva de perfil de desempenho manteve-se sempre acima daquela proporcionada pelo outro critério.

6.3.4 Hill climbing.

Concluimos os testes realizados no Matlab investigando a influência do uso do *hill climbing* tanto sobre o tempo de processamento do algoritmo genético como sobre a qualidade da solução por este obtida. Essa análise visa avaliar se esse procedimento, que requer um elevado tempo de processamento, produz resultados que justifiquem sua utilização.

Lembramos que esse procedimento usa a técnica denominada *2-interchange* para explorar a vizinhança de uma solução. Essa técnica, cuja complexidade é da ordem de n^2 , é geralmente empregada até que não exista um indivíduo vizinho à solução atual com solução melhor que

Classe de problemas		Maior distância	Menor final da janela
C1	Acumulado	8376,8	8626,2
	Média	930,76	958,46
C2	Acumulado	8326,5	9196,2
	Média	1040,81	1149,53
R1	Acumulado	16297,7	17833,2
	Média	1358,14	1486,10
R2	Acumulado	12725,5	14120,7
	Média	1156,87	1283,70
RC1	Acumulado	12286,1	13341,4
	Média	1535,77	1667,68
RC2	Acumulado	10751,6	12063,6
	Média	1343,95	1507,95
Acumulado Total		68764,2	75181,3
Média Total		1227,93	1342,52

Tabela 6.3: Comparação do critério de escolha do cliente inicial de cada rota na construção da solução inicial pelo método PFIH

esta. Entretanto, para evitar que algoritmo genético consumisse um tempo excessivo nessa busca, optamos por limitar a procura por vizinhos, interrompendo o *hill climbing* após 50 iterações, mesmo que a última solução encontrada não seja um mínimo local.

Três alternativas foram analisadas: a supressão do *hill climbing*, o uso constante dessa estratégia e o seu emprego a cada 10 iterações do algoritmo. No caso em que o *hill climbing* foi aplicado em todas as iterações, também variamos a probabilidade de sua aplicação sobre os cromossomos existentes na população obtida ao final de uma iteração (ou geração).

A Tabela 6.4 resume os resultados obtidos. Nesta tabela, o termo *Prob. Hill* indica a probabilidade de aplicação do *hill climbing* a cada cromossomo da população. Um valor igual a 0,5, por exemplo, indica que cada cromossomo tem 50% de chances de ser selecionado para

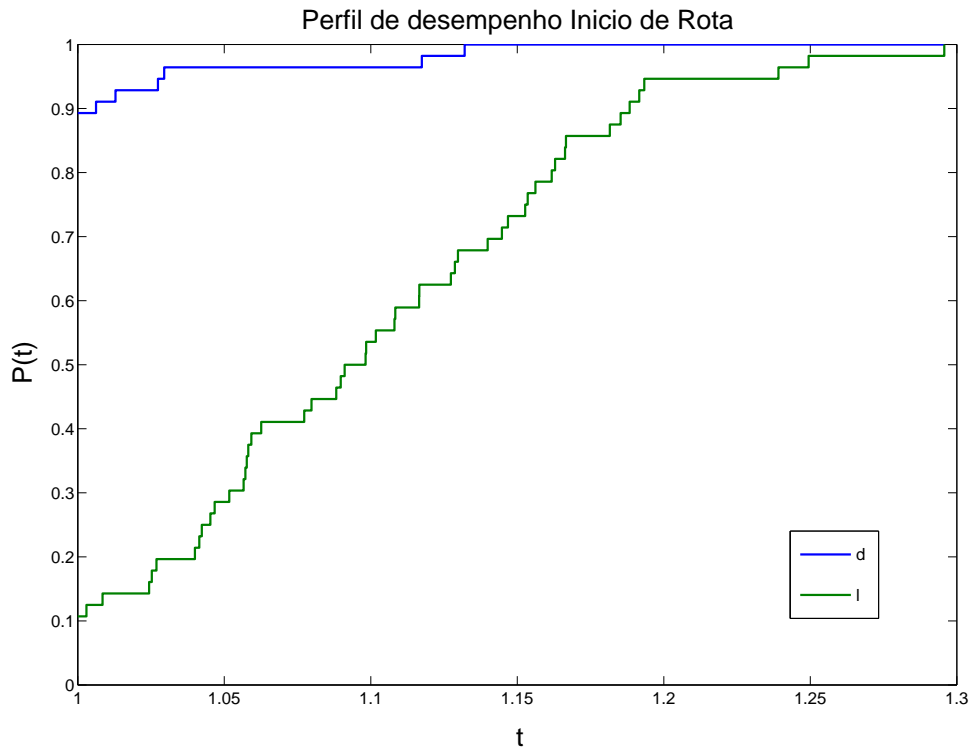


Figura 6.3: Qualidade da solução obtida para cada tipo de cliente inicial das rotas do algoritmo PFIH. A letra d simboliza o cliente mais distante do depósito, enquanto a letra l representa o cliente com o menor instante final da janela de tempo.

a aplicação do *hill climbing*. A coluna *Tempo* fornece o tempo total consumido pelo algoritmo, em segundos.

Neste experimento, variamos o número de iterações do algoritmo genético, para verificar se um número maior de iterações produziria solução comparável àquela obtida pelo *hill climbing*. No caso em que este não foi aplicado, pudemos executar um número elevado de iterações. Por outro lado, quando o procedimento foi usado, o tempo gasto pelo algoritmo cresceu bastante e foi preciso limitar as iterações.

Na Tabela 6.4, os valores acumulados e médios das soluções obtidas são fornecidos em três colunas. A coluna denominada “distância” contém as distâncias totais (acumuladas e médias) obtidas. Já a coluna “nv” fornece o número total acumulado e o número médio de veículos utilizados na solução. Finalmente, a última coluna da tabela apresenta os tempos de processamento do algoritmo, em segundos.

		Prob. Hill	Iterações	Distância	nv	Tempo (s)
Sem Hill Climbing	Acumulado	0	500	67735,0	654	2639,7
	Média	-		1209,55	11,7	47,1
	Acumulado	0	1000	67489,0	660	5166,4
	Média	-		1205,16	11,8	92,3
Com Hill Climbing	Acumulado	0,5	50	66276,4	647	93819,9
	Média			1183,51	11,6	1675,4
	Acumulado	0,25	100	66636,8	651	86388,7
	Média			1189,94	11,6	1542,7
Hill Climbing a cada 10 iter.	Acumulado	0,5	100	66724,8	643	9567,8
	Média			1191,51	11,5	170,9

Tabela 6.4: Influência do uso do *hill climbing* sobre a distância total encontrada, o número de veículos utilizados e o tempo de processamento do algoritmo genético

Observando a Tabela 6.4, notamos que a implementação sem o *hill climbing* exigiu um tempo de processamento pequeno, mesmo quando 1000 iterações foram executadas. Por outro lado, a duplicação do número de iterações provocou um aumento compatível do tempo de processamento, sem provocar, entretanto, uma grande melhora da solução.

A Figura 6.4 fornece os perfis de desempenho relativos às combinações de probabilidade de aplicação do *hill climbing* e de número total de iterações do algoritmo analisadas em nossos testes. Neste gráfico, os termos “sem hill 500” e “sem hill 1000” referem-se, respectivamente, ao algoritmo que não usa o *hill climbing* e para o qual se estabeleceu um limite de 500 e 1000 iterações. Da mesma forma, os termos “com hill 50” e “com hill 100” referem-se aos testes feitos com o uso de *hill climbing*, com um limite de 50 iterações e 100 iterações, conforme descrito na Tabela 6.4. Finalmente, o termo “hill a cada 10 iter” refere-se ao algoritmo em que 100 iterações foram executadas, com a aplicação do *hill climbing* a cada 10 iterações.

Pelo gráfico da Figura 6.4, constatamos que o algoritmo sem *hill climbing* fornece soluções com qualidade inferior. E embora o aumento do número de iterações provoque uma melhora da qualidade da solução obtida, essa melhora não é suficiente para que se atinja o desempenho

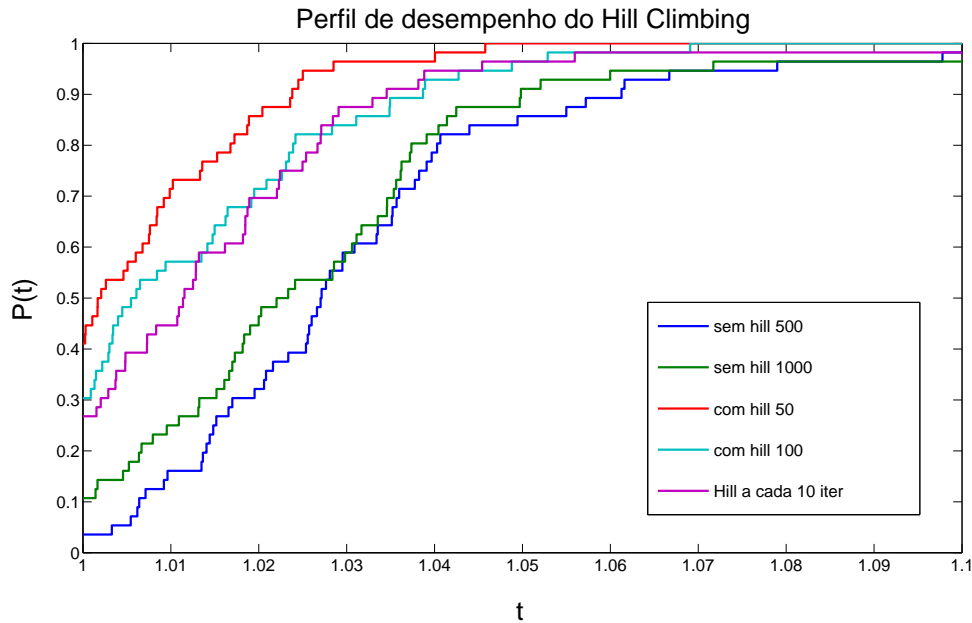


Figura 6.4: Qualidade da solução obtida combinando valores diferentes da probabilidade de aplicação do hill climbing e do número de iterações do algoritmo genético.

das versões do algoritmo que usam o *hill climbing*.

Obtivemos perfis de desempenho muito semelhantes ao usarmos *hill climbing* a cada 10 iterações e ao usarmos esse procedimento em todas as iterações, mas com uma probabilidade menor de aplicação a cada cromossomo da população. Apesar das soluções obtidas por essa última alternativa terem se mostrado levemente superiores, ela apresentou um tempo médio cerca de 800% superior à primeira.

As melhores soluções foram aquelas obtidas definindo a probabilidade de *hill climbing* em 0,5 e o número de iterações em 50. Como se observa na Figura 6.4, o perfil de desempenho dessa combinação manteve-se sempre acima dos perfis das demais alternativas. Entretanto, o tempo de processamento do algoritmo foi bastante elevado neste caso. Ainda assim, optamos por adotar o valor 0,5 para a probabilidade de *hill climbing*.

6.4 Testes com a implementação em c++.

Em virtude do elevado tempo de processamento do programa desenvolvido no Matlab, que provocou a adoção de um limite baixo para o número de iterações, uma versão do algoritmo

genético foi implementada em c++. Nesta seção, discutimos os resultados gerados por essa nova implementação quando aplicada aos problemas de Solomon [102], além de compará-la ao algoritmo em Matlab e cotejá-la com outros algoritmos descritos na literatura.

Com exceção das mudanças inerentes à conversão de um programa escrito em uma linguagem procedural para outra orientada a objetos, os algoritmos escritos em matlab e em c++ são equivalentes. Assim, por exemplo, os parâmetros ora utilizados são aqueles que propiciaram os melhores resultados nos testes descritos na Seção 6.3.

Entretanto, uma pequena alteração foi feita na rotina que executa o *hill climbing*. Como o novo algoritmo é mais rápido do que aquele implantado no matlab, em lugar de restringirmos em 50 o número máximo de indivíduos explorados, interrompemos o procedimento quando a diferença entre a distância total alcançada pelo melhor indivíduo encontrado na vizinhança da solução atual e a distância total obtida por esta não supera um fator ϵ . Em outras palavras, o *hill climbing* pára quando

$$| d(S_k) - \min\{d(S) : S \in V(S_k)\} | < \epsilon,$$

onde S_k é a solução atual, $V(S_k)$ é a sua vizinhança, e $d(S)$ é a distância total fornecida por uma solução S . Essa alteração evita o emprego de um procedimento caro em troca de uma melhora discreta.

6.4.1 Comparação com o algoritmo desenvolvido no Matlab.

A Tabela 6.5 mostra os resultados obtidos pelas implementações em matlab e em c++. Para esse experimento, o número de iterações foi fixado em 50 e o fator ϵ do *hill climbing*, usado no c++, assumiu o valor 0,005.

As distâncias totais obtidas para os 56 problemas indicam uma superioridade da implementação em c++, que gerou soluções melhores. A Figura 6.5 corrobora essa análise, evidenciando de forma indiscutível a superioridade da versão em c++.

A Figura 6.6 mostra que a implementação em c++ foi, em geral, a mais rápida. Observando os perfis de desempenho, constatamos que o algoritmo em Matlab consumiu ao menos o dobro do tempo da versão em c++ em mais de 40% dos problemas. Além disso, em cerca de 1/3 dos casos, o algoritmo em matlab foi no mínimo três vezes mais lento que o outro.

Classe de problemas		Matlab			c++		
		Distância	nv	Tempo (s)	Distância	nv	Tempo (s)
C1	Acumulado	7914,3	93	12140,6	7501,1	90	4077,0
	Média	879,36	10,3	1349,0	833,45	10,0	453,0
R1	Acumulado	15868,5	180	22922,3	14403,8	167	8713,0
	Média	1322,37	15,0	1910,2	1200,31	13,9	726,1
RC1	Acumulado	11920,0	116	14615,0	11136,4	109	4646,0
	Média	1490,00	14,5	1826,9	1392,05	13,6	580,8
C2	Acumulado	8096,4	82	11119,7	7511,3	80	10352,0
	Média	1012,05	10,3	1390,0	938,91	10,0	1294,0
R2	Acumulado	12128,3	97	18958,8	10606,8	95	21195,0
	Média	1102,57	8,8	1723,5	964,25	8,6	1926,8
RC2	Acumulado	10348,9	79	14063,5	9213,8	79	10615,0
	Média	1293,61	9,9	1757,9	1151,73	9,9	1326,9
Acumulado Total		66276,4	647	93819,9	60373,1	620	59598,0
Média Total		1183,51	11,6	1675,4	1078,09	11,1	1064,3

Tabela 6.5: Desempenho comparado das implementações em Matlab e em c++

Entretanto, uma análise cuidadosa da tabela e dos gráficos mostra que, para os problemas com janela de tempo mais longa, os valores relativos ao tempo de processamento mantiveram-se próximos, tendo o programa criado no Matlab superado a implementação em c++ em alguns problemas da classe R2. Essa vantagem do Matlab foi provocada pelo uso excessivo do *hill climbing* pelo programa em c++. Uma vez que a regra de parada desse procedimento não envolve um limite de iterações, ele é muito executado quando há algum progresso na exploração da vizinhança da solução corrente. Naturalmente, este aumento do tempo é compensado por uma redução expressiva da distância total obtida pelo algoritmo genético. Assim, para a classe R2, as soluções obtidas pelo programa em Matlab são cerca de 14% inferiores àquelas encontradas pela implementação em c++. Para as demais classes, a diferença média está próxima de 9%.

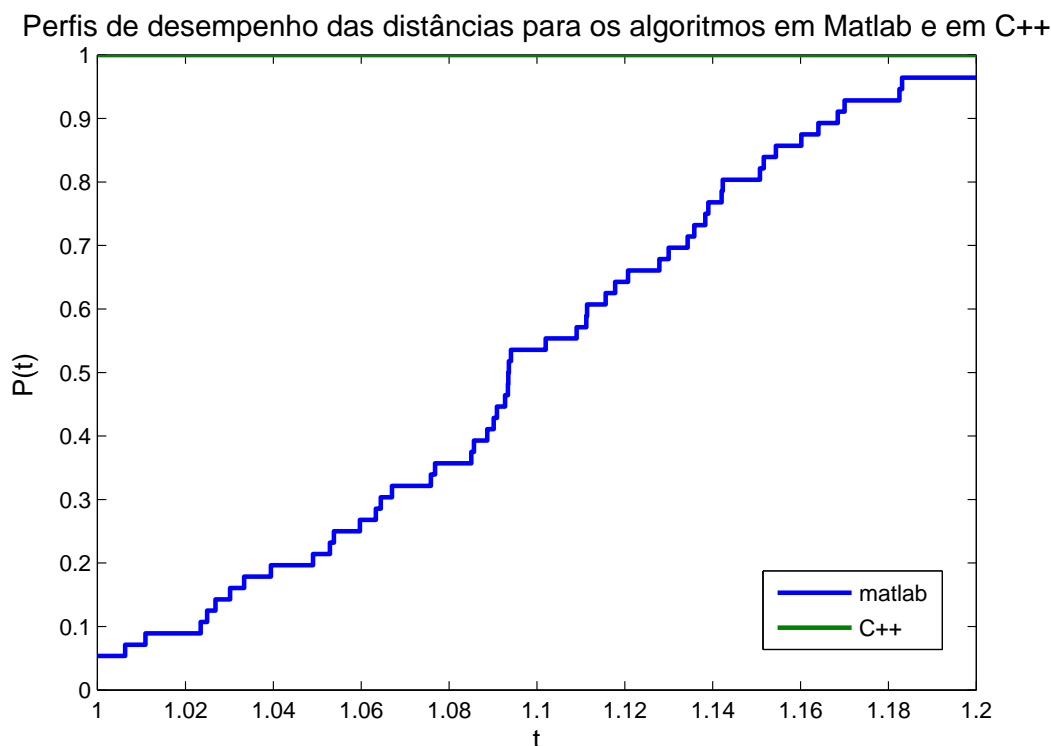


Figura 6.5: Perfis de desempenho das distâncias obtidas pelas implementações em Matlab e c++.

6.4.2 Ponderação entre distância percorrida e número de veículos.

Até o momento, preocupamo-nos em comparar valores diferentes para os parâmetros do algoritmo genético. Por essa razão, optamos por manter fixa a capacidade dos veículos para as 56 instâncias, contrapondo o que ocorre na maioria dos artigos dedicados ao tema. Nestes artigos, os problemas pertencentes às classes do Grupo 1 possuem veículos com capacidade igual a 200. Já no Grupo C2, essa capacidade é aumentada para 700, enquanto nos Grupos R2 e RC2 a capacidade é de 1000. Uma vez que os testes exibidos na próxima seção têm por finalidade verificar a eficiência do nosso algoritmo quando comparado com outros disponíveis na literatura, doravante adotaremos essas novas capacidades.

Outra alteração adotada a partir dessa seção é a exclusão dos problemas da Classe C1. Essa exclusão se deve ao fato de que tanto o nosso algoritmo como a grande maioria dos métodos existentes chegam à solução ótima desses problemas sem grandes esforços computacionais. Assim,

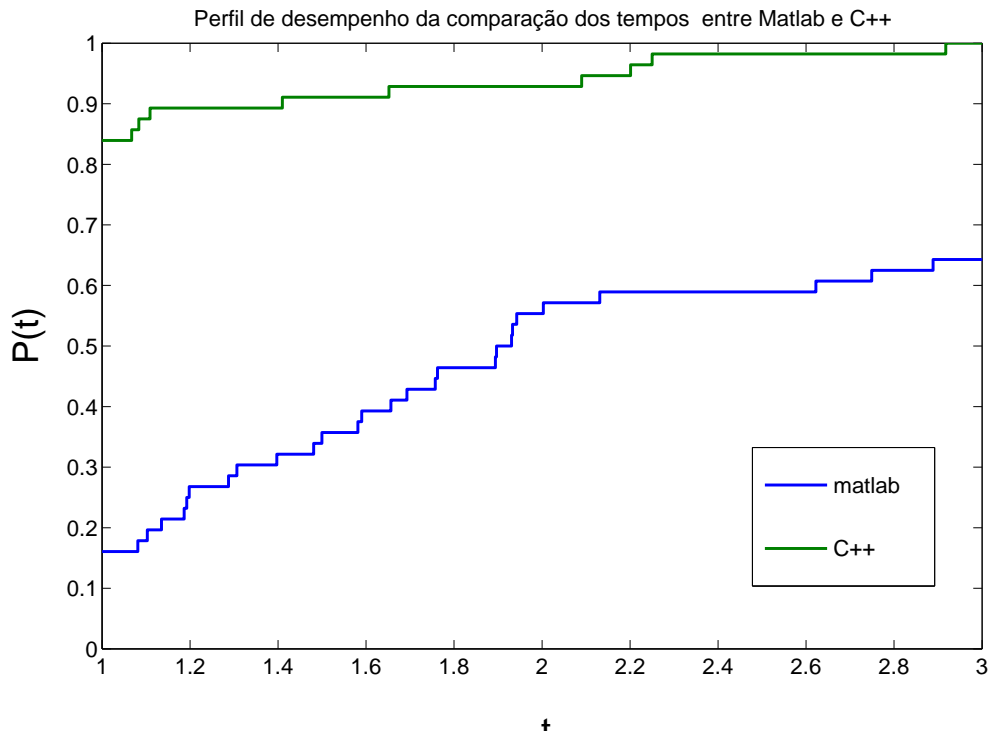


Figura 6.6: Perfis de desempenho do tempo gasto pelas implementações em Matlab e c++.

como essa classe não oferece desafios dignos de nota, reduzimos a 47 o número de problemas que utilizamos.

Em nosso trabalho, optamos por definir como objetivo principal a minimização da distância percorrida. No entanto, a maioria dos textos encontrados na literatura utiliza como objetivo principal a minimização do número de veículos. Para permitir uma análise que englobe esses dois aspectos, alteramos nossa função objetivo, ponderando o números de veículos e a distância total utilizada. A nova função é definida por

$$f(S) = p_{dist}d(S) + p_{nv}K(S),$$

em que $d(S)$ e $K(S)$ são, respectivamente, a distância total e o número de veículos empregados na solução S , e os parâmetros $p_{dist} > 0$ e $p_{nv} > 0$ são constantes que permitem a ponderação dos dois objetivos.

Uma comparação entre diferentes combinações dos pesos p_{dist} e p_{nv} pode ser vista na Tabela 6.6. Nesta tabela, a coluna “nv” contém o número acumulado total ou o número médio de

Classe de problemas		$p_{dist} = 1$ $p_{nv} = 0$		$p_{dist} = 1$ $p_{nv} = 2000$		$p_{dist} = 1$ $p_{nv} = 1000$		$p_{dist} = 0$ $p_{nv} = 1$	
		Distância	nv	Distância	nv	Distância	nv	Distância	nv
R1	Acum.	14538,2	170	14527,3	168	14535,6	170	17308,0	171
	Média	1211,51	14,2	1210,61	14,0	1211,30	14,2	1442,34	14,3
RC1	Acum.	11215,2	112	11228,2	110	11259,6	112	13107,7	112
	Média	1401,90	14,0	1403,52	13,8	1407,45	14,0	1638,46	14,0
C2	Acum.	4932,2	28	5049,7	29	4965,0	28	6069,70	30
	Média	616,53	3,5	631,21	3,6	620,63	3,5	758,71	3,8
R2	Acum.	10270,8	78	10300,2	74	10269,3	75	14688,0	42
	Média	933,71	7,1	936,38	6,7	933,58	6,8	1335,27	3,8
RC2	Acum.	8520,1	61	8534,0	60	8528,0	61	13472,4	33
	Média	1065,01	7,6	1066,74	7,5	1066,01	7,6	1684,06	4,1
Acum. Total		49476,5	449	49639,3	441	49557,6	446	64645,8	388
Média Total		1052,69	9,6	1056,16	9,4	1054,42	9,5	1375,44	8,3

Tabela 6.6: Comparação entre os pesos da função objetivo

veículos das soluções. As Figuras 6.7 e 6.8 contêm, respectivamente, os perfis de desempenho associados à distância total e ao número de veículos.

Em geral, considerando os 47 problemas, o algoritmo que produz as menores distâncias totais é aquele no qual $p_{nv} = 0$. Da mesma forma, quando analisamos apenas o número de veículos utilizados, a combinação que produz os melhores resultados é aquela na qual $p_{dist} = 0$ e $p_{nv} = 1$.

Observando a Figura 6.7, reparamos que o desempenho do algoritmo quando $p_{dist} = 0$ é bastante fraco. Já as demais combinações de p_{dist} e p_{nv} produzem resultados similares. Em particular, todas obtiveram a menor distância para um número próximo de um terço dos problemas. Apenas com o aumento do valor de t notamos uma vantagem clara do uso de $p_{nv} = 0$.

O gráfico da Figura 6.8 mostra resultado compatível. Agora, a combinação $p_{dist} = 0$ e $p_{nv} = 1$ é claramente a melhor, enquanto as outras têm comportamento semelhante. Ainda

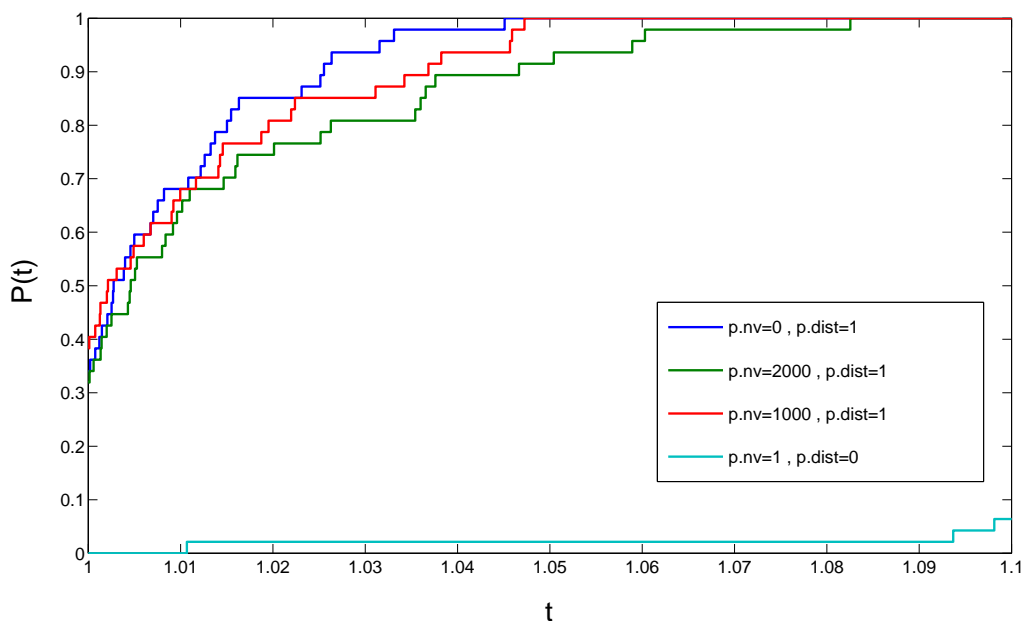


Figura 6.7: Perfis de desempenho com base na distância total.

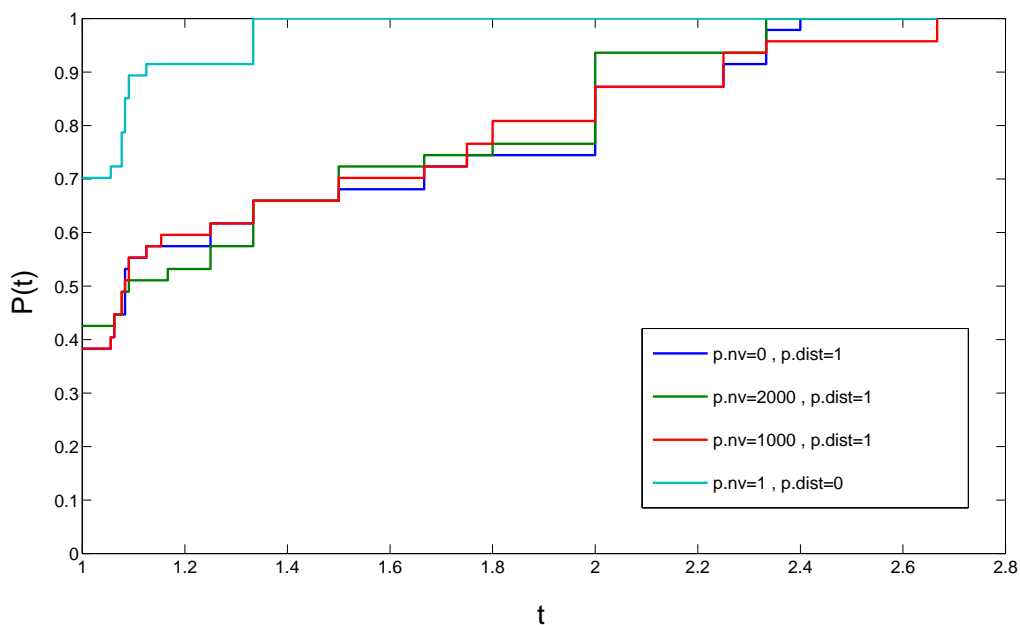


Figura 6.8: Perfis de desempenho com base no número de veículos.

assim, até mesmo a opção de definir $p_{nv} = 0$ fornece a melhor solução em mais de 30% dos problemas.

6.4.3 Comparação com outros algoritmos.

Terminando esse capítulo, vamos comparar nosso algoritmo genético a outras meta-heurísticas desenvolvidas para o problema de roteamento com janela de tempo. Nos testes apresentados abaixo, adotamos dois critérios de parada do algoritmo. Em uma primeira versão, paramos quando a função objetivo não diminui em 50 iterações consecutivas. Na segunda versão, o algoritmo é interrompido após 1000 iterações. Em ambos os casos, utilizamos os melhores parâmetros encontrados nos testes anteriores. Como nossa ênfase foi a minimização da soma das distâncias das rotas, adotamos $p_{dist} = 1$ e $p_{nv} = 0$.

Uma vez que muitos algoritmos foram propostos na literatura e seria impossível comparar nosso programa a todos, extraímos do artigo de Bräysy e Gendreau [25], que comparara diversos algoritmos baseados em meta-heurísticas e heurísticas populares, os dois métodos que, aparentemente, fornecem os melhores resultados. O primeiro algoritmo, proposto por Bräysy [20] (vide página 38), foi o que apresentou, em geral, o menor número de número de veículos. O segundo algoritmo, criado por Jung e Moon [68], forneceu as menores distâncias totais.

Um resumo dos resultados apresentados pelos algoritmos mencionados acima pode ser encontrado na Tabela 6.7.

Como se observa na tabela, dentre as opções de critério de parada que analisamos, o algoritmo que faz 1000 iterações obteve distâncias totais menores para todas as cinco classes de problemas testadas. Esse resultado é confirmado pelo perfil de desempenho dado na Figura 6.9. Quando avaliamos o número de veículos utilizados, os resultados também favorecem a versão do algoritmo na qual paramos após 1000 iterações, embora a diferença não tenha sido expressiva para várias classes de problemas.

Expandindo a análise aos resultados obtidos na literatura, notamos que o algoritmo que apresentamos aqui encontrou soluções com um número maior de veículos. Esse comportamento já era esperado, uma vez que não tínhamos como objetivo reduzir o número de veículos utilizados. No tocante às distâncias totais, nosso algoritmo obteve valores intermediários, que superam os alcançados pelo algoritmo que valoriza apenas o número de veículos, mas que são inferiores aos da meta-heurística que privilegia a distância. Considerando, entretanto, que o algoritmo de Jung e Moon [68] só pára após repetir 2000 iterações sem que haja mudança na solução

Classes de problemas		50 iterações		1000 iterações		Bräysy		Jung e Moon	
		Distância	nv	Distância	nv	Distância	nv	Distância	nv
R1	Acum.	14412,4	167	14261,4	162	14665,4	143	14159,4	159
	Média	1201,03	13,9	1188,45	13,5	1222,12	11,9	1179,95	13,3
RC1	Acum.	11047,5	108	10904,4	107	11116,6	92	10749,1	104
	Média	1380,93	13,5	1363,05	13,4	1389,58	11,5	1343,64	13,0
C2	Acum.	4764,2	24	4718,9	24	4718,9	24	4718,9	24
	Média	595,53	3,0	589,86	3,0	589,86	3,0	589,86	3,0
R2	Acum.	10102,7	71	9855,9	66	10726,3	30	9662,5	59
	Média	918,42	6,5	896,00	6,0	975,12	2,7	878,41	5,4
RC2	Acum.	8309,2	57	8149,4	56	9027,0	26	8033,7	50
	Média	1038,64	7,1	1018,68	7,7	1128,38	3,3	1004,21	6,3
Acum. Total		48636,0	427	47889,9	415	50254,3	315	47323,6	396
Média Total		1034,81	9,1	1018,93	8,8	1069,24	6,7	1006,88	8,4

Tabela 6.7: Distância total e número de veículos para o algoritmo genético desenvolvido neste trabalho e para as meta-heurísticas selecionadas na literatura

ótima, enquanto executamos apenas 1000 iterações no total, podemos dizer que as soluções que obtivemos são bastante satisfatórias.

Para a classe C2 em particular, nosso algoritmo teve um bom desempenho tanto em relação ao número de veículos quanto à distância total. Nesta classe, aliás, os algoritmos apresentaram resultados bastante similares.

Nas Tabelas 6.8 a 6.12, comparamos as melhores soluções obtidas pelo algoritmo desenvolvido nesse trabalho aos melhores resultados baseados em heurísticas e meta-heurísticas fornecidas na literatura especializada (extraídos do sítio [120]) e aos resultados ótimos encontrados pelos algoritmos exatos apresentados no artigo de Alvarenga *et al.* [2]. Os campos em branco correspondem a problemas para os quais não foi possível encontrar a solução exata.

Como era de esperar, os resultados expostos nas Tabelas 6.8 a 6.12 indicam que as distâncias obtidas por nosso algoritmo são menores que aquelas apresentadas pelos melhores meta-heurís-

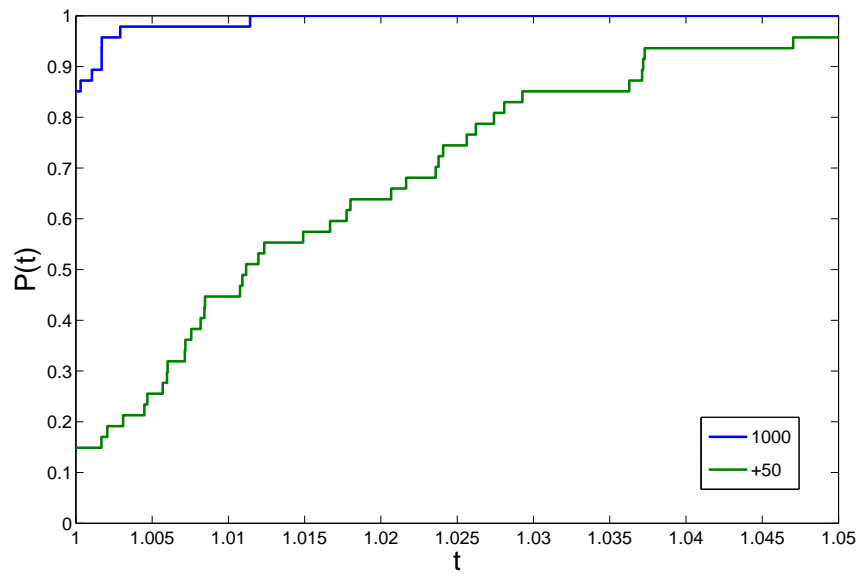


Figura 6.9: Perfis de desempenho (relativos às distâncias totais) para os dois critérios de parada utilizados.

ticas disponíveis na literatura. Isso se deve ao fato dessas meta-heurísticas usarem, em geral, o número de veículos como objetivo principal. Em compensação, nosso algoritmo exige sempre uma frota maior, naturalmente.

Usando como termo de comparação as soluções exatas fornecidas por Alvarenga *et al.* [2], constatamos que o número de veículos das soluções obtidas por nosso algoritmo está sempre muito próximo do valor ótimo. Já a distância que obtemos é relativamente pior. Acreditamos que essa diferença poderia ser reduzida se adotássemos um número maior de iterações.

Prob.	Nosso Algoritmo		Metaheurísticas		Algoritmos Exatos	
	Distância	nv	Distância	nv	Distância	nv
r101	1642,9	20	1645,8	19	1637,7	20
r102	1475,7	18	1486,1	17	1466,6	18
r103	1222,1	15	1292,7	13	1208,7	14
r104	992,6	12	1007,2	9		
r105	1360,8	15	1377,1	14	1355,3	15
r106	1243,0	13	1252,0	12	1234,6	13
r107	1085,2	12	1104,7	10	1064,6	11
r108	951,6	10	960,9	9		
r109	1157,7	13	1194,7	11	1146,9	13
r110	1083,4	12	1118,6	10	1068	12
r111	1061,5	12	1096,7	10	1048,7	12
r112	970,0	11	982,1	9		

Tabela 6.8: Comparação entre os melhores resultados produzidos nesse trabalho e os obtidos na literatura para a classe R1.

Prob.	Nosso Algoritmo		Metaheurísticas		Algoritmos Exatos	
	Distância	nv	Distância	nv	Distância	nv
rc101	1650,6	16	1696,9	14	1619,8	15
rc102	1480,5	15	1554,8	12	1457,4	14
rc103	1332,1	13	1261,7	11	1258,0	11
rc104	1151,0	10	1135,5	10		
rc105	1543,5	16	1629,4	13	1513,7	15
rc106	1396,1	13	1424,7	11		
rc107	1212,1	12	1230,5	11		
rc108	1134,9	11	1139,8	10		

Tabela 6.9: Comparação entre os melhores resultados produzidos nesse trabalho e os obtidos na literatura para a classe RC1.

Prob.	Nosso Algoritmo		Metaheurísticas		Algoritmos Exatos	
	Distância	nv	Distância	nv	Distância	nv
c201	591,6	3	591,6	3	589,1	3
c202	591,6	3	591,6	3	589,1	3
c203	591,2	3	591,2	3	588,7	3
c204	596,6	3	590,6	3		
c205	588,9	3	588,9	3	586,4	3
c206	588,5	3	588,5	3	586,0	3
c207	588,3	3	588,3	3	585,8	3
c208	588,3	3	588,3	3	585,8	3

Tabela 6.10: Comparação entre os melhores resultados produzidos nesse trabalho e os obtidos na literatura para a classe C2.

Prob.	Nosso Algoritmo		Metaheurísticas		Algoritmos Exatos	
	Distância	nv	Distância	nv	Distância	nv
r201	1153,7	9	1252,4	4	1143,2	8
r202	1039,7	7	1191,7	3		
r203	877,2	6	939,5	3		
r204	743,8	4	825,5	2		
r205	970,7	7	994,4	3		
r206	890,7	5	906,1	3		
r207	827,4	5	890,6	2		
r208	738,5	3	726,8	2		
r209	887,4	6	909,2	3		
r210	926,4	7	939,3	3		
r211	780,9	5	892,7	2		

Tabela 6.11: Comparação entre os melhores resultados produzidos nesse trabalho e os obtidos na literatura para a classe R2.

Prob.	Nosso Algoritmo		Metaheurísticas		Algoritmos Exatos	
	Distância	nv	Distância	nv	Distância	nv
rc201	1269,0	9	1406,9	4	1261,8	9
rc202	1099,5	8	1365,7	3	1092,3	8
rc203	943,4	6	1049,6	3		
rc204	810,6	4	798,4	3		
rc205	1175,7	9	1297,2	4	1154,0	7
rc206	1064,2	7	1146,3	3		
rc207	980,5	7	1061,1	3		
rc208	785,7	5	828,1	3		

Tabela 6.12: Comparação entre os melhores resultados produzidos nesse trabalho e os obtidos na literatura para a classe RC2.

Considerações Finais

O problema de roteamento de veículos com janela de tempo tem grande importância no bom gerenciamento da distribuição de determinados produtos e serviços pelas empresas. Uma vez que o PRVJT exige um alto esforço computacional, pois pertence à classe dos problemas NP-difíceis, torna-se importante uma boa escolha do método usado para sua solução.

Neste trabalho, apresentamos as mais importantes técnicas de solução do PRVJT, incluindo desde os métodos exatos até as mais promissoras meta-heurísticas desenvolvidas nos últimos tempos. Essas meta-heurísticas, que combinam diferentes métodos de busca e empregam procedimentos específicos de refinamento e de criação de soluções iniciais de alta qualidade, são os algoritmos que, hoje em dia, fornecem resultados superiores, segundo Braÿsy *et al.* [25, 23].

Para resolver o problema de roteamento com janela de tempo, elaboramos um algoritmo genético que combina

- a representação dos cromossomos por números inteiros;
- as técnicas de pós-otimização *Hill Climbing* e Recuperação;
- a geração de parte da população inicial através de uma heurística de inserção baseada no trabalho de Solomon [101];
- o uso de diversas estratégias de cruzamento.

Para definir um algoritmo genético eficiente, diferentes combinações de técnicas e parâmetros foram formadas e aplicadas ao conjunto de problemas criado por Solomon [102]. A comparação entre o algoritmo desenvolvido nesse trabalho e os melhores métodos encontrados na literatura evidenciou a qualidade de nosso algoritmo, que forneceu soluções próximas às aquelas obtidas pelos demais métodos.

A partir de nossos experimentos computacionais, foi possível concluir que o uso da estratégia PMX para a realização do cruzamento se mostrou mais eficiente que o cruzamento heurístico e o cruzamento por fusão, não havendo diferenças muito significativas entre esses dois tipos de cruzamento.

Os experimentos também revelaram a importância da utilização de populações iniciais formadas por indivíduos oriundos da vizinhança da solução inicial construída com o uso do método PFIH. Isto reforça a idéia de que é necessário incluir um conjunto de bons indivíduos na população, para que estes passem suas boas características às próximas gerações. No que se refere à implementação do método PFIH, investigamos alternativas para a definição do nó inicial de cada rota. Como mencionamos no Capítulo 6, para os problemas criados por Solomon [102], a escolha do cliente mais distante do depósito como nó inicial das rotas conduziu o algoritmo genético a um desempenho melhor.

Pudemos observar, ainda, que o uso do *hill climbing* produz resultados de qualidade superior, o que evidencia a necessidade do emprego de métodos que refinam as soluções obtidas pelo algoritmo genético. Como essa técnica exige um esforço computacional significativamente alto, podemos usá-la em apenas algumas iterações, de modo que a relação entre o tempo computacional e a qualidade da solução seja aceitável. Outra forma de controlar o tempo consumido pelo *hill climbing*, consiste em limitar o número de iterações do método, ou estabelecer um critério de parada baseado na melhoria obtida para a função objetivo do problema.

Os resultados obtidos ponderando-se, na função objetivo, a distância total das rotas com o número de veículos nos mostraram que o algoritmo genético que desenvolvemos não é tão competitivo em relação aos publicados na literatura quando privilegiamos apenas o número de veículos. Esse mau resultado é decorrência de termos dedicado todos os procedimentos do algoritmo à minimização da distância total. Assim, a simples mudança da função objetivo não foi suficiente para a obtenção de um programa eficiente. Por outro lado, a minimização

combinada da distância total e do número de veículos, com pesos $p_{dist} = 1$ e $p_{nv} = 1000$, mostrou-se eficiente para os problemas das classes R1 e R2, superando os resultados obtidos quando privilegiamos apenas a distância (ou, seja, quando $p_{nv} = 0$).

Dentre os possíveis desdobramentos desse trabalho, podemos citar a análise de outras técnicas de refinamento e de obtenção de uma solução inicial, com o objetivo de produzir novos tipos de vizinhança e uma maior diversidade de bons indivíduos. Além disso, também seria interessante adaptar o algoritmo desenvolvido à minimização do número de veículos. Finalmente, seria conveniente estudar estratégias diferentes para a divisão do cromossomo em rotas, analisando, por exemplo, a possibilidade de que o cromossomo seja percorrido ao contrário, iniciando o processo pela alocação do último gene ao último veículo.

Referências Bibliográficas

- [1] Aarts, J.; Korst, H. M. e Van Laarhaven, P. J. M. Simulated annealing. Em *Local Search in Combinatorial Optimization*. E. Aarts, J. K. Lenstra, eds. John Wiley and Sons, Chichester, UK, p.91-120, 1997.
- [2] Alvarenga, G.B.; Mateus, G.R. e De Tomi, G. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & Operations Research*, 34, p.1561-1584, 2007.
- [3] Atkinson, J. B. A greedy look-ahead heuristic for combinatorial optimisation: An application to vehicle scheduling with time windows. *J. Operational. Research Society*, 45(6), p.673-684, 1994.
- [4] Aytug, H.; Bhattacharyya, S. e Koehler, G.J. A Markov chain analysis of general cardinality genetic algorithms with power of cardinality alphabets. *European Journal of Operational Research*, 96(1), p.195-201, 1996.
- [5] Aytug, H. e Koehler, G.J. Stopping criteria for finite length genetic algorithms. *ORSA Journal on Computing*, 8(2), p.183-191, 1996.
- [6] Badeau, P; Gendreau, M; Guertin, F.; Potvin. J-Y e Taillard, E. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research*, 5, p.109-122, 1997.
- [7] Baker, E.K. Vehicle routing with time windows constraints. *Logistic and Transportation Review*, 18(4), p.385-401, 1982.

- [8] Baker, J.E. Reducing bias and inefficiency in the selection algorithm. Em *Proceedings of the Second International Conference on Genetic Algorithms and their Application* (Hillsdale), p.14-21, 1987.
- [9] Bard, J.F.; Kontoravdis,G.; Yu,G. A Branch and Cut Procedure for the Vehicle Routing Problems with Time Windows. *Transportation Science*, 36(2), p.250-269, 2002.
- [10] Barrie M. Baker, M. A. Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5), p.787-800, 2003.
- [11] Bent, R., P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4), p.515-530, 2004.
- [12] Benyania, I. e Potvin, J. Generalization and Refinement of Route Construction Heuristic using Genetic Algorithms. Em *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*. Piscataway: IEEE Service Center, p.39-43, 1995.
- [13] Berger, J.; Salois, M.; Begin, R. A Hybrid Genetic Algorithms for The Vehicle Routing Problem with Time Windows. Em *Lecture Notes in Artificial Intelligence* 1418, p. 114-127, 1998.
- [14] Berger, J., M. Barkaoui, O. Bräysy. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *Inform. Systems Operational Research*, 41, p.179-194, 2003.
- [15] Bramel, J. e Simchi-Levi, D. Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operational Research*, 44, p.501-509, 1993.
- [16] Le Bouthillier, A.; Crainic,T. G. e Keller, R . K. Co-operative Parallel Method for Vehicle Routing Problems with Time Windows. Em *4th Metaheuristics International Conference*, presented at MIC Porto, Portugal, 2001.
- [17] Bräysy, O. A Hybrid Genetic Algorithms for the Vehicle Routing Problem with Time Windows. Licentiate thesis, University of Vaasa, Finland, 1999.

- [18] Bräysy, O. A new Algorithms for Vehicle Routing Problem with Time Windows based on the Hybridization of a Genetic Algorithms and Route Construction Heuristic. Em *Proceedings of the University of Vaasa*, Research papers 227, 1999.
- [19] Bräysy, O.; Berger, J. e Barkaoui, M. A New Hybrid Evolutionary Algorithms fo the Vehicle Routing Problem with Time Windows. Em *Route 2000 workshop*, Skodsborg, Denmark. 2000.
- [20] Bräysy, O. A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows. *Inform Journal on computing*, 15(4), p. 347-368, 2003.
- [21] Bräysy, O.; Berger, J.; Barkaoui, M. e Dullaert, W. A Threshold Accepting Metaheuristic for the Vehicle Routing Problem with Time Windows. *Central European. J. Operational Research*, 11(4), p.369-387, 2003.
- [22] Bräysy, O.; Dullaert, W. e Gendreau, M. Evolutionary Algorithms for the Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 10(6), p.587-611, 2004.
- [23] Bräysy, O.; Hasle, G. e Dullaert, W. A multi-start local search algorithm for the vehicle routing problem with time windows. *European Journal of Operational Research*, 159(3), p. 586-605, 2004.
- [24] Bräysy , O. e Gendreau,M. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1), p.104-118, 2005.
- [25] Bräysy , O. e Gendreau,M. Vehicle routing problem with time windows, Part II: Metaheuristics *Transportation Science*,39(1), p. 119-139, 2005.
- [26] Bullnheimer,B.; Hartl,R. F. e Strauss, C. Applying the Ant System to the Vehicle Routing Problem. Em *2nd International Conference on Metaheuristics*, Sophia-Antipolis, France. 1997.
- [27] Chiang, W. C.; Russell,R. A. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Ann.Operations Research*, 63, p.3-27, 1996.

- [28] Chu, P. e Beasley, J. A Genetic Algorithm for the Generalised Assignment Problem. *Computers and Operations Research*, 24, pp. 17-23, 1997.
- [29] Clarke, G. e Wright, J.W. Scheduling of Vehicles from a Central Depot to a number of Delivery Points. *Operations Research*, 12, p.568-581, 1964.
- [30] Colomi, A.; Dorigo, M. e Maniezzo. Distributed Optimization by ant Colonies. Em *Proceedings of the European Conference on Artificial Life*. F. Varela and P. Bourguine, eds, Elsevier, Amsterdam, Holanda. 1991.
- [31] Cook, W. e Rich, J.L. A Parallel Cutting-Plane Algorithm for the Vehicle Routing Problems with Time Windows. Working Paper, Department of Computational and Applied Mathematics Rice University, Houston, U.S.A. 1999.
- [32] Cordone, R. and Calvo, R.W. A Heuristic for the Vehicle Routing Problem with Time Windows. *Journal of Heuristic* 7, p. 107-129, 2001.
- [33] Dantzig, G. B. e Ramser, R. H. The Truck Dispatching Problem. *Management Science*, 6, p.80-91, 1959.
- [34] Dantzig, G. B. e Wolfe, P. The decomposition algorithm for linear programming. *Operations Research*, 8, p.101-111, 1960.
- [35] Davis, L. Applying Adaptive Algorithms to Epistatic Domains. Proceedings of the International Joint Conference on Artificial Intelligence, p.162-164, 1985.
- [36] Desrochers, M.; Lenstra, J.K.; Savelsbergh, M.W.P. e Soumis, F. Vehicle Routing with Time Windows: Optimization and Approximation. Em *Vehicle Routing: Methods and Studies*. B. Golden and A. Assad(eds.). Elsevier Science Publishers. Amsterdam, Holanda , p.65-84, 1988.
- [37] Desrochers, M.; Desrosiers, J. e Solomon, M. A new optimization algorithms for Vehicle Routing Problem with Time Windows. *Operations Research*, 40(2), p.342-354, 1992.

- [38] Desrosiers, J.; Dumas, Y.; Solomon, M.M. e Soumis, F. Time Constrained Routing and Scheduling. Em *Handbooks in Operations Research and Management Science 8: Network Routing*. M.O Ball, T.L. Magnanti, C.L.Monma, G.L.Nemhauser(eds.). Elsevier Science Publishers. Amsterdam, Holanda , p.35-139, 1995.
- [39] Dolan, E.D. e Moré,J.J. Benchmarking optimization software with performance profiles. *Math. Program.*, A91, p.201-213, 2002.
- [40] Dueck,G. e Scheurer, T. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90, p.161-175, 1990.
- [41] Feo, T.A. e Resende, M.G.C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, p.109-133, 1995.
- [42] Fischer, M. Vehicle routing. Em *Handbooks in operations research and management science, v.8: network routing*. M.O. Ball *et al.* (eds). Amsterdam, Elsevier, p.1-33, 1995.
- [43] Fogel, L.J. Autonomous Automata. *Industrial Research*, 4, p.14-19, 1962.
- [44] Gambardella L.M.; Taillard, E. e Agazzi, G. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. Em *New Ideas in Optimization*. D. Corne, M. Dorigo and F. Glover (eds), McGraw-Hill, p.63-76, 1999.
- [45] Garcia, B.L.; Potvin, J.Y. e Rousseau, J.M. A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Windows Constraints. *Computers & Operational Research*, 21, p.1025-1033, 1994.
- [46] Garey, M.R. e Johnson, D.S. *Computers and intractability: a guide to the theory of NP-completeness*. New York, W. H. Freeman, 1979.
- [47] Gehring, H. e Homberger, J. A Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problems with Time Windows. Em *Proceedings of Eurogen99*. Jyväskylä: University of Jyväskylä, p.57-64, 1999.

- [48] Gehring, H. e Homberger, J. Parallelization of Two-Phase Metaheuristic for Routing Problems with Time Windows. *Asia-Pacific Journal of Operational Research*, 18, p.35-47, 2001.
- [49] Gendreau, M.; Hertz, A. e Laporte, G. A new insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40, p.1086-1093, 1992.
- [50] Gillett, B.; e Miller, L.R. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22, p.340-349, 1974.
- [51] Glover, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13, p.533-549, 1986.
- [52] Glover, F. Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Working paper, College of Business and Administration, University of Colorado, Boulder, CO. 1991.
- [53] Glover, F. New ejection chain and alternating path methods for traveling salesman problems. O. Balci, R. Sharda, S. Zenios(eds). *Computer Science and Operations Research: New Developments in Their Interfaces*. Pergamon Press, Oxford, U.K., p.449-509, 1992.
- [54] Golden, B.L. e Assad, A. A. Perspectives on Vehicle Routing: Exciting New Developments. *Operations Research*, 35, p.6-17, 1986.
- [55] Golden, B.L. e Assad, A. A. *Vehicle Routing: Methods and Studies*. Elsevier Science Publishers, Amsterdam. 1988.
- [56] Goldberg D.E. e Lingle R. Alleles, loci, and the TSP. Em *Proceedings of the First International Conference on Genetic Algorithms*, illsdale, NJ, p.154-159, 1985.
- [57] Goldberg, D.E. *Genetic Algorithms in Search. Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [58] Greenhalgh, D. e Marshall, S. Convergence criteria for genetic algorithms. *Journal of Computing*, 30, p.269-282, 2000.

- [59] Grefenstette, J.J. Incorporation Problem Specific Knowledge into Genetic Algorithms. Em *Genetic Algorithms and Simulated Annealing*, L. Davis(Ed.), Morgan Kaufmann Publishers, San Mateo, p.42-60, 1987.
- [60] Grefenstette, J.J. e Baker, J. How Genetic Algorithms Work: A Critical look at Implicit Parellelism. Em *Proceedings of the third International Conference on Genetic Algorithms*, p.75-91, 1989.
- [61] Hansen, P. e Mladenovic, N. Variable Neighborhood Search: Methods and Recent Applications. Em *Proceedings of the Third Metaheuristic International Conference*, p. 275-280, 1999.
- [62] Holland, J.H. *Adaptation in Natural and Artificial System*. University of Michigan Press, Ann Arbor, 1975.
- [63] Homberger, J. e Gehring, H. Two Evolutionary metaheuristic for the Vehicle Routing Problems with Time Windows. *Infoms Systems Operational Research*, 37, p.297-318, 1999.
- [64] Homberger, J. e Gehring, H. A two-phase hybrid metaheuristic for the Vehicle Routing Problem with Time Windows. *European Journal of Operational Research*, 162, p. 220-238, 2005.
- [65] Hwang, H. S. An Improved Model for Vehicle Routing Problem with Time Constraint based on Genetic Algorithm. *Computers & Industrial Engineering*, 42, p.361-369, 2002.
- [66] Ioannou, G.; Kritikos, M. e Prastacos, G. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *J. Operations Research Society*, 52, p.523-537, 2001.
- [67] Jepsen, M.; Petersen, B.; Spoorendonk, S. e Pisinger, D. A non-robust branch-and-cut-and-price algorithm for the vehicle routing problem with time windows. Technical Report, Department of Computer Science, University of Copenhagen, Denmark. 2006
- [68] Jung, S. e Moon, B.R. A hybrid Genetic Algorithms for the Vehicle Routing Problem with Time Windows. Em *Proceedings of Genetic and Evolutionary Computation Conference*. San Francisco: Morgan Kaufmann Publishers, p.1309-1316, 2002.

- [69] Kallehauge, B. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7), p.2307-2330, 2008.
- [70] Kilby, P.; Prosser, P. e Shaw, P. Guided local search for the vehicle routing problem with time windows. Em *META-HEURISTICS Advanced Trends Local Search Paradigms for Optimization*. S. Voss, S. Martello, I. H. Osman, C. Roucairol, eds. Kluwer Academic Publishers, Boston, MA, 473-486, 1999.
- [71] Kirkpatrick, S.; Gellat, C.D. e Vecchi, M.P. Optimization by simulated annealing. *Science*, 220, p.671-680, 1983.
- [72] Kolen, A.W.J.; Rinnooy, A.H.G. e Trienekens, H.W.J.M. Vehicle Routing with Time Windows. *Operations Research*, 35(2), p.266-273, 1987.
- [73] Kontoravdis, G. A. e Bard, J. F. A GRASP for the vehicle routing problem with time windows. *INFORMS J. on Computing*, 7(1), p.10-23, 1995.
- [74] Lau, H.C.; Sim, M. e Teo, K.M. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148, p.559-569, 2003.
- [75] Lenstra, J. e Rinnooy Kan, A. Complexity of vehicle routing and scheduling problems. *Networks*, 11, p.221-227, 1981.
- [76] Lin, S. Computer solutions of the traveling salesman problem. *J. Bell System Technical*, 44, p.2245-2269, 1965.
- [77] Lin, S. e Kernighan, B. "An Effective Heuristic Algorithm for the Traveling Salesman Problem", *Operations Research*, 21, p.498-516, 1973.
- [78] Louis, S. J.; Yin, X. e Yuan, Z. Y. Multiple Vehicle Routing with time windows using Genetic algorithms. *Evolutionary Computation*, 3, p.1804-1808, 1999.
- [79] Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H. e Teller, E. Equation of state calculation by fast computing machines. *J. of Chemical. Physics*. 21, p.1087-1091, 1953.

- [80] Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd, Springer. 1996.
- [81] Mladenovic, N. e Hansen, P. Variable neighborhood search. *Computers and Operations Research*, 24, p.1097-1100, 1997.
- [82] Nix, A. e Vose, M.D. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5, p.79-88, 1992. 79-88.
- [83] Ochi, L.S.; Vianna, D.S.; Drummond, L.M.A. e Victor, A.O. An Evolutionary Hybrid Metaheuristic for Solving the Vehicle Routing Problem with Heterogeneous Fleet. *Lecture Notes in Computer Science*, 1391, p.216-224, 1998.
- [84] Or, I. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. Ph.D. thesis, Northwestern University, Evanston, IL. 1976.
- [85] Osman I.H. e Christofides N. Simulated Annealing and descent algorithms for capacitated clustering problem. Research Report, Imperial College, University of London. 1989.
- [86] Osman, I. H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Ann. Operations Research*, 41, p.421-452, 1993.
- [87] Potvin, J.Y.; Rousseau, J.M. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European. J. Operations. Research*, 66, p.331-340, 1993.
- [88] Potvin, J.Y. e Rousseau, J.M. An Exchange Heuristic for Routing Problems with Time Windows. *Journal of the Operational Research Society*, 46, p.1433-1446, 1995.
- [89] Potvin, J.Y. e Bengio, S. The Vehicle Routing Problem with Time Windows - Part II: Genetic Search. *Inform Journal on Computing*, 8, p.165-172, 1996.
- [90] Rechenberg, I. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Stuttgart, Frommann-Holzboog, 1973.

- [91] Reeves, C.R. *Modern Heuristic Techniques for Combinatorial Problems*, London, McGraw-Hill, 1995.
- [92] Rego C. A subpath ejection method for the vehicle routing problem. *Management Science*, 44, p.1447-1459, 1998.
- [93] Rochat, Y. e Taillard, E. D. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1), p.147-167, 1995.
- [94] Rousseau, L.M.; Gendreau, M. e Pesant, G. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8, p.43-58, 2002.
- [95] Rudolph, G. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5(1), p.96-101, 1994.
- [96] Russell, R. An effective heuristic for the M-tour traveling salesman problem with some side conditions. *Operations Research*, 25, p.517-524, 1977.
- [97] Russell, R. A. Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29, p.156-166, 1995.
- [98] Schulze J. e Fahle, T. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86, p.585-607, 1999.
- [99] Schwefel, H-P. *Evolutionstrategie und Numerische Optimierung*, Tese de doutorado, Technical University of Berlin, 1975.
- [100] Shaw, P. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. *Lecture Notes in Computer Science*, 1520, p.417-431, 1998.
- [101] Solomon, M.M. Algorithms for vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2), p.254-266, 1987.
- [102] Solomon, M.M. e Desrochers, J. Time Windows Constrained Routing and Scheduling Problems. *Transportation Science*, 22, p.1-13, 1988.

- [103] Sontrop, H.M.J.; Horn, S.P.; van der and Teeuwen, G e Uetz,M. Fast Ejection chain algorithms for vehicle routing with time windows. Research memoranda 012, Maastricht Research School of Economics of Technology and Organization, Maastricht, NL, 2005.
- [104] Taillard, É.; Badeau, P.; Gendreau, M.; Guertin, F. e Potvin, J.-Y. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31, p.170-186, 1997.
- [105] Tan K.C.; Lee L.H.; Zhu Q.L. e Ou k. Heuristic methods for vehicle problem with time windows. *Artificial Inteligence in Engineering*, 15, p. 281-295, 2000.
- [106] Thangiah, S.; Osman, I. e Sun, T. Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows. Working Paper, Institute of Mathematics and Statistics, University of Kent, UK, 1994.
- [107] Thangiah, S.R. Vehicle Routing with Time Windows using Genetic Algorithms. Em: *Application Handbook of Genetic Algorithms: New Frontiers*, v.2, L. Chambers Ed.. CRC, p.253-277, 1995.
- [108] Thangiah,S.R. An Adaptive Clustering Method Using a Geometric Shape for Vehicle Routing Problems with Time Windows. In *Proceedings of 6th International Coference on Genetic Algorithms*. San Francisco: Morgan Kaufmann, p.536-543, 1995.
- [109] Thompson, P.M. Local Search algorithms for Vehicle Routing and Other Combinatorial Problems. Ph.D. Thesis, 1988. MIT, Cambridge, Mass.
- [110] Thompson, P.M. and Orlin, J.B. Theory of Cyclic Transfer. Working paper, Operations Research Center, MIT, Cambridge, Mass. 1989.
- [111] Thompson, P. M. e Psaraftis, H.N. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41, p.935-946, 1993.
- [112] Toth, P. Vigo, D. *The vehicle routing problem* . SIAM, 2002.
- [113] Van Landeghem, H. R. G. A bi-criteria heuristic for the vehicle routing problem with time windows. *European. J. Operational Research*, 36, p.217-226, 1988.

- [114] Vose, M.D. e Lielins, G.E. Punctuated equilibria in genetic search. *Complex Systems*, 5, p.31-44, 1991.
- [115] Voudouris, C. Guided local search for combinatorial problems. Ph.D. thesis, 1997. Department of Computer Science, University of Essex, Colchester, UK.
- [116] Voudouris, C. e Tsang, E. Guided local search. *European. J. Operational Research*, 113, p.80-119, 1998.
- [117] Whitley, D.; Starkweather, T. e Shaner, D. The traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, Handbook of Genetic Algorithms, Van Nostrand Reinhold, NY. 1991.
- [118] Zhu, K.Q. A New Genetic Algorithms for the Vehicle Routing Problem with Time Windows. Presented at IC-AI, Las Vegas, U.S.A. 2000.
- [119] Zhu, K.Q. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. Tools with Artificial Intelligence, Em *Proceedings. 15th IEEE International Conference*, p.176-183, 2003.
- [120] www.sintef.no/static/am/opti/projects/top/vrp/bknown.html. Visitado em novembro de 2008.