

**UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**



**UTILIZAÇÃO DE ALGORITMOS GENÉTICOS
HÍBRIDOS E WEB 2.0 NO DESENVOLVIMENTO
DE SISTEMAS DE ROTEIRIZAÇÃO DE
VEÍCULOS**

Rodrigo César de Oliveira

Dezembro

2008

Dados Internacionais de Catalogação na Publicação (CIP)

- O48u Oliveira, Rodrigo César de, 1982-
 Utilização de algoritmos genéticos híbridos e Web 2.0 no desenvolvimento de sistemas de roteirização de veículos / Rodrigo César de Oliveira. - 2009.
 75 f. : il.
- Orientador: Keiji Yamanaka.
 Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
 Inclui bibliografia.
1. Algoritmos genéticos - Teses. I. Yamanaka, Keiji. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 681.3.06

**UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA
ELÉTRICA**

**UTILIZAÇÃO DE ALGORITMOS GENÉTICOS
HÍBRIDOS E WEB 2.0 NO DESENVOLVIMENTO
DE SISTEMAS DE ROTEIRIZAÇÃO DE
VEÍCULOS**

Rodrigo César de Oliveira

Texto da dissertação apresentada à
Universidade Federal de Uberlândia,
perante a banca de examinadores
abaixo, como parte dos requisitos
necessários à obtenção do título de
Mestre em Ciências. Aprovada em 05
de dezembro de 2008.

Banca examinadora:

Keiji Yamanaka, Ph.D. - Orientador (UFU)
Alexsandro dos Santos Soares, Dr. (UFG)
Luciano Vieira Lima, Dr. (UFU)

UTILIZAÇÃO DE ALGORITMOS GENÉTICOS HÍBRIDOS E WEB 2.0 NO DESENVOLVIMENTO DE SISTEMAS DE ROTEIRIZAÇÃO DE VEÍCULOS

Rodrigo César de Oliveira

Texto da dissertação apresentada à Universidade Federal de Uberlândia como parte dos requisitos para obtenção do título de Mestre em Ciências.

Prof. Keiji Yamanaka, Ph.D.
Orientador

Prof. Darizon Alves de Andrade, Ph.D.
Coordenador do curso de Pós-Graduação

“A insatisfação é o primeiro passo para o progresso de um homem ou de uma nação.”
Oscar Wilde

Agradecimentos

Agradeço primeiramente a Deus que me presenteou com vida, capacidade de obter conhecimento e inspiração para concluir esse trabalho.

Ao meu pai Ângelo, minha mãe Nilta e minha irmã Pollyanne, minha família e meu alicerce, que proporcionaram condições para minha formação como pessoa e minha vida acadêmica e principalmente por todo o apoio e carinho que recebi deles.

À minha namorada Poliana, uma pessoa muito importante pra mim, que esteve comigo nos momentos mais difíceis e foi a principal incentivadora para que eu concluísse esse trabalho.

Ao meu orientador, Prof. Keiji, pela compreensão, paciência e também pela confiança que depositou em mim em todos os trabalhos que desenvolvemos juntos.

E finalmente, a todos os meus amigos, cujos nomes são muitos para serem enumerados, que me ajudaram de forma direta ou indireta.

Resumo

A utilização de um sistema computacional de roteirização de veículos é imprescindível para a competitividade de empresas que trabalham com distribuição de recursos e produtos. Apesar de demandar um alto custo, esse tipo de sistema oferece um retorno garantido do investimento feito.

Utilizando-se as técnicas de Metaheurísticas combinadas com os recursos oferecidos livremente pela *web* é possível desenvolver sistemas de roteirização com funcionalidades, desempenho e eficiência bem próximos a sistemas comerciais.

Nesse trabalho, primeiramente foi realizado um estudo dos fundamentos teóricos dos Algoritmos Genéticos. Utilizando-se a linguagem de programação Java, implementou-se um conjunto de rotinas comuns para os AGs, sob forma de uma biblioteca de desenvolvimento. De posse dessas rotinas, foi desenvolvido um algoritmo de roteirização de veículos com suporte a múltiplas rotas e diversos tipos de cálculos de custos entre localidades. Para a melhoria do desempenho do algoritmo implementado, utilizou-se a técnica de busca local 2opt, resultando-se assim em um Algoritmo Genético Híbrido.

Esse algoritmo foi utilizado como núcleo para o desenvolvimento de um protótipo funcional de um sistema *web* de roteirização de veículos capaz de manipular as coordenadas geográficas de localidades na superfície terrestre e gerar um mapa com as rotas de veículos considerando as vias de circulação veiculares. O sistema gera também uma lista detalhada das rotas e descrevendo minuciosamente a direção a ser tomada em cada trecho, facilitando a atuação do condutor do veículo.

Palavras-chave

Algoritmos Genéticos Híbridos, roteirização, otimização.

Abstract

The use of a computer system for vehicle routing is essential for the competitiveness of companies working with distribution of resources and products. Despite high cost, such a system offers a guaranteed return on investment made.

Using a Metaheuristics techniques combined with the resources currently offered by web, free development tools can be developed routing systems with features, performance and efficiency very close to commercial systems.

In this work, was first performed a study of the theoretical foundations of Genetic Algorithms. Using the Java programming language, is implemented a set of routines common to the AGs, in the form of a library of development. With this routines, was developed an algorithm routes of vehicles with support for multiple routes and various types of calculations of costs between locations. To improve the performance of the algorithm implemented, we used the technique of local search 2opt, and are thus resulting in a Hybrid Genetic Algorithm.

This algorithm was used as a core for the development of a functional prototype of a web vehicles routing system capable of handling the geographical coordinates of settlements in land surface and create routes on a map of vehicles considering the vehicular traffic routes . The system also generates a detailed list of routes and close to describing direction to be taken on each site to facilitate the work of a driver of the vehicle.

Keywords

Hybrid Genetic Algorithms, routing, optimization

Sumário

Lista de Figuras	xi
Lista de Tabelas	xii
Simbologia	xiii
1 Introdução	1
1.1 O problema da otimização de rotas	1
1.2 Sistemas de Roteirização de Veículos	2
1.3 O problema do caixeiro viajante	5
1.4 Objetivos do Trabalho	6
2 Algoritmos Genéticos	7
2.1 Introdução	7
2.2 Aplicações	8
2.3 Representação Cromossomial	9
2.3.1 Representação binária	9
2.3.2 Representação por inteiros	10
2.3.3 Representação por reais	11
2.4 Função de Aptidão	11
2.5 Seleção	11
2.5.1 Método da Roleta	11
2.5.2 Ranking	12
2.5.3 Torneio	13
2.6 Cruzamento	13
2.6.1 Cruzamento de um ponto	13
2.6.2 Cruzamento de múltiplos pontos	14
2.6.3 Partially-Mapped Crossover(PMX)	14
2.6.4 Edge Recombination Crossover(ERX)	16
2.6.5 Cycle Crossover(CX)	17

2.6.6	Order Crossover(OX)	17
2.6.7	Order Based Crossover(OX2)	19
2.7	Mutação	20
2.7.1	Exchange Mutation(EM)	20
2.7.2	Simple Inversion Mutation(SIM)	20
2.7.3	Insertion Mutation(ISM)	21
2.7.4	Inversion Mutation(IVM)	21
2.7.5	Scramble Mutation(SM)	22
2.8	Fundamentos Matemáticos	22
2.9	AGs Híbridos	24
3	Implementação de um Sistema de Roteirização de Veículos	26
3.1	Introdução	26
3.2	Tecnologias Utilizadas	26
3.2.1	Linguagem de Programação Java	26
3.2.2	AJAX e Google Web Toolkit	28
3.2.3	Google Maps API	29
3.2.4	MySQL	29
3.3	Arquitetura do Sistema	31
3.4	Banco de Dados	31
3.5	Implementação do AG	34
3.6	Implementação da Aplicação <i>web</i>	37
4	Testes e Resultados	42
4.1	Introdução	42
4.2	Teste do AG	42
4.3	Teste do Sistema	45
4.4	Considerações	55
5	Conclusões e Trabalhos Futuros	56
5.1	Conclusões	56
5.2	Publicações	58
5.3	Trabalhos Futuros	58
	Referências Bibliográficas	60

Lista de Figuras

2.1	<i>Gráfico de $f(x) = (x^2 + x)\text{sen}(x)$</i>	9
2.2	<i>Cromossomos na roleta</i>	12
2.3	<i>Cromossomos pais com genes mapeados para o PMX</i>	15
2.4	<i>Replicação dos genes mapeados nos filhos</i>	15
2.5	<i>Finalização do operador PMX</i>	16
2.6	<i>CX - Primeiro ciclo</i>	18
2.7	<i>CX - Segundo ciclo</i>	18
2.8	<i>Exemplo de EM</i>	20
2.9	<i>Exemplo de SIM</i>	21
2.10	<i>Exemplo de ISM</i>	21
2.11	<i>Exemplo de IVM</i>	21
2.12	<i>Exemplo de SM</i>	22
3.1	<i>Comparação da popularidade do Java com outras linguagens</i>	27
3.2	<i>Tela inicial do Google Maps</i>	30
3.3	<i>Arquitetura do sistema</i>	32
3.4	<i>Diagrama de Entidade-Relacionamento do banco de dados</i>	34
3.5	<i>Coordenadas geográficas na esfera</i>	36
3.6	<i>Exemplo do operador $2opt$</i>	37
3.7	<i>Cadastro de localidade</i>	38
3.8	<i>Exemplo de busca das coordenadas de localidade</i>	39
3.9	<i>Exemplo de rota gerada</i>	41
4.1	<i>Rota gerada pelo AG</i>	44
4.2	<i>Gráfico de Desempenho do AG</i>	45
4.3	<i>Rota gerada para o veículo 1</i>	53
4.4	<i>Rota gerada para o veículo 2</i>	54

Lista de Tabelas

1.1	<i>Exemplos de utilização de Sistemas de Roteirização</i>	4
2.1	<i>Exemplos de cromossomos binários</i>	10
4.1	<i>Coordenadas no plano cartesiano</i>	43
4.2	<i>Agências dos Correios</i>	46

Simbologia

AGs – Algoritmos Genéticos
 P_c – Probabilidade de cruzamento
 P_m – Probabilidade de mutação
TSP – *Travel Salesman Problem*
JVM – *Java Virtual Machine*
TCP – *Transmission Control Protocol*
IP – *Internet Protocol*
HTTP – *Hypertext Transfer Protocol*
FTP – *File Transfer Protocol*
API – *Application Programming Interface*
HTML – *HyperText Markup Language*
XML – *eXtensible Markup Language*
AJAX – *Asynchronous Javascript And XML*
GWT – *Google Web Toolkit*
RIA – *Rich Internet Application*
SDK – *Software Development Kit*
JDK – *Java Development Kit*
SQL – *Structured Query Language*
DER – Diagrama de Entidade-Relacionamento
GHz – Giga Hertz
GB – Giga Bytes
ms – milisegundos

Capítulo 1

Introdução

1.1 O problema da otimização de rotas

Com o crescimento dos centros urbanos e popularização do comércio eletrônico, a logística tornou-se um campo chave para a sobrevivência da maioria das empresas. Com o aumento da concorrência, os clientes estão cada vez mais exigentes que suas entregas sejam *in full* e *on time*, ou seja, querem qualidade na entrega e dentro dos prazos estipulados [1]. Como exemplos da aplicação da logística, pode-se citar:

- entrega em domicílio de produtos comprados em um supermercado ou uma loja virtual;
- distribuição de bebidas em bares e restaurantes;
- entrega de correspondência e encomendas pelos Correios;
- distribuição de produtos de um atacadista para lojas varejistas;
- coleta de lixo urbano.

Atualmente, de um lado, tem-se um fluxo cada vez maior de entregas, um conjunto cada vez mais rígido de restrições e um tempo de entrega estabelecido cada vez menor. Do outro, tem-se computadores com grande capacidade de processamento e armazenamento.

Isso faz com que a utilização de sistemas computacionais de roteirização seja a maneira mais eficiente da logística atingir seus objetivos.

O termo roteirização não é encontrado em dicionários da língua portuguesa, mas vem sendo utilizado como equivalente ao termo *routing* da língua inglesa. Entende-se por roteirização o processo de determinação de uma ou mais rotas ou seqüências de paradas a serem cumpridos por veículos de uma frota, objetivando visitar um conjunto de pontos previamente determinados, e geralmente, geograficamente distribuídos.

A logística não se restringe somente ao transporte. Ela também gerencia outras áreas, tais como movimentação e armazenagem de produtos, processamento de pedidos e gerenciamento de informações, que não serão tratadas nesse trabalho.

1.2 Sistemas de Roteirização de Veículos

Os sistemas de roteirização de veículos executam, de forma automatizada, tarefas que seriam humanamente inviáveis. São capazes de considerar variáveis como distância percorrida, tempo de trânsito, capacidade de carga, horários de atendimento, prazos de entregas, disponibilidade da frota, tempos de paradas, entre outras. Alguns são capazes de traçar as rotas sobre mapas e até mesmo serem integrados com sistemas de localização, possibilitando que os veículos sejam monitorados em tempo real.

A evolução desses sistemas ao longo do tempo acompanhou a evolução da computação e vem desde a fase em que eram executados em *mainframes* onde não dispunham de interface gráfica e requeriam um altíssimo tempo de processamento, até os dias de hoje que dispõe-se de avançados recursos tanto de *hardware* quanto de *software*. Eles são compostos basicamente de um algoritmo de otimização que envolve cálculos matemáticos e heurísticas e alguma forma de visualização das rotas, geralmente mapas.

Atualmente, estão disponíveis no mercado várias opções de sistemas de roteirizações com diversas características e funcionalidades, como o Trucks, RouteSmart e TrucksStops [2, 3, 4]. Já existem alguns sistemas desenvolvidos no Brasil sendo comercializados, como é o caso do BRinCity & BRonRoad [5]. Porém ainda predominam os sistemas importados.

Geralmente demandam um alto investimento para serem adquiridos e implantados, mas o retorno é garantido quando comparados a utilização de métodos manuais de determinação de rotas. Na tabela 1.1 são citados alguns exemplos do impacto da adoção desses sistemas em algumas empresas [6].

Tabela 1.1: *Exemplos de utilização de Sistemas de Roteirização*

Empresa	Ramo	Problema	Sistema	Resultados
Protege	Transporte de valores	Otimização de rota e frota	Trucks 9.0	15% de redução da frota, mais rigor nos horários, maior qualidade de serviço
Sadia	Distribuição de alimentos	Otimização de rota	Trucks 8.2	Redução de 5% do custo operacional
Adriano Coselli	Atacadista	Redução no tempo de distribuição e Km	Trucks 8.2	Redução de 2h no tempo total e duplicação das entregas (500 p/ semestre)
Luiz Tonin	Atacadista	Redução no tempo de distribuição e Km	Trucks 8.2	Ganho de 3 Km p/ entrega, 30 entregas diárias p/ veículo, redução de 20% no tempo de entrega e ampliação da área de atuação
Cofesa	Atacadista	Redução no tempo de distribuição e Km	Trucks 8.2	Rotas mais “exatas”, maior controle da frota
Grupo Benjamin	Atacadista	Redução no tempo de distribuição e Km	Trucks 8.2	Passou a atender 350 pedidos diários
Marilan Biscoitos	Distribuição de alimentos	Redução do tempo e custo de distribuição	Trucks	Redução de 5% nos custos de entrega, ganhos em produtividade, eficiência, qualidade e confiança dos clientes
Fuller	Distribuição de alimentos	Redução no tempo de distribuição e Km	Trucks	Economia de 27% nos custos de venda e agilização das vendas
Grupo Martins	Distribuição de atacado	Rapidez no atendimento	Trucks	Redução no tempo de ciclo e agibilidade na entrega
Panamco Spal	Distribuição de bebidas	Otimização de frota	RoadShow	Redução de 25% no n° de veículos, maior eficiência na entrega
Jardim América	Distribuição de bebidas	Redução de custos, otimização frota/rotas	RouteSmart	Redução de 10% a 15% do custo de distribuição, otimização de rotas
Kibon	Distribuição de sorvetes	Agilizar distribuição	TruckStops	Mais de 4000 clientes em 2 meses, redução de 25% em Km e 95% na ocupação dos veículos

1.3 O problema do caixeiro viajante

Os problemas de roteirização são derivados de um problema clássico de otimização combinatória denominado problema do caixeiro viajante (*Traveling Salesman Problem - TSP*) que vem sendo estudado desde 1800 em vários campos da ciência, como Matemática, Engenharia e Computação [7].

O problema consiste em um caixeiro viajante que deseja visitar um determinado número de cidades. Ele deve passar somente uma vez por cada uma e retornar à cidade de origem percorrendo a menor distância possível.

Embora aparentemente simples, é um problema altamente complexo pertencente à classe dos problemas Não Polinomiais Difíceis (*Non polynomial Hard - NP-Hard*) [8].

Existem várias maneiras de se obter soluções aproximadas para esse problema. A mais imediata que se poderia pensar seria traçar todas as rotas possíveis e verificar qual delas resultou em uma menor distância percorrida. Porém essa estratégia é inviável devido ao grande número de combinações possíveis.

Uma outra maneira, seria através da utilização de alguma metaheurística. As metaheurísticas são métodos heurísticos genéricos, que servem para resolver vários tipos de problemas de otimização, onde não se conhece algum algoritmo exato eficiente. A seguir tem-se alguns exemplos de metaheurísticas:

- Algoritmos Genéticos
- Colônia de Formigas
- *Simulated Annealing*
- Busca TABU
- GRASP
- *Particle Swarm Optimization*

Apesar de não oferecerem garantia de uma solução ótima, as metaheurísticas vêm sendo largamente utilizadas e obtendo bons resultados tanto no TSP quanto em outros problemas de otimização combinatorial.

1.4 Objetivos do Trabalho

Esse trabalho tem como objetivo aplicar conceitos teóricos da Inteligência Artificial para a resolução de problemas do mundo real. Primeiramente, será feito um estudo da teoria dos Algoritmos Genéticos, e, em seguida, eles serão utilizados na implementação de um sistema de roteirização de veículos.

Não será implementado um sistema completo, e sim um protótipo funcional contemplando as funcionalidades mais importantes de um sistema de roteirização e o mais próximo possível de um sistema comercial. O sistema deverá gerar rotas otimizadas considerando a distância e o tempo de entrega. Deverá suportar rotas múltiplas e ainda otimizar a utilização de veículos, dado um número de veículos disponíveis. Na realização dos testes, serão processadas localidades reais, considerando os valores de suas coordenadas geográficas (latitude e longitude).

O sistema deverá possuir interface gráfica com o usuário e ser capaz de traçar rotas sobre o mapa e, se possível, considerar rodovias, ruas e avenidas para o tráfego dos veículos. Além de ser possível a visualização das rotas, é desejável que ele seja capaz de descrever as rotas com detalhes para facilitar a orientação do condutor do veículo.

Para o desenvolvimento desse trabalho, optou-se por utilizar apenas linguagens de programação, servidores, banco de dados, ferramentas e *frameworks* que se enquadrem na categoria de software livre. Com isso pretende-se demonstrar que com um pouco de pesquisa e alguns recursos de software disponíveis gratuitamente, é possível gerar sistemas com funcionalidades similares a sistemas comerciais de alto custo, acessíveis somente a grandes corporações.

Capítulo 2

Algoritmos Genéticos

2.1 Introdução

Algoritmos Genéticos (AGs) são algoritmos de busca e otimização inspirados na Teoria da Seleção Natural e na Genética. É um método de uso geral que pode ser aplicado a vários tipos de problemas.

Os AGs foram desenvolvidos por volta de 1960 por John Holland. Em 1975, ele publicou o livro *Adaptation in Natural and Artificial Systems*. Os estudos de Holland foram continuados por seu aluno David Goldberg que, nos anos 80, conseguiu realizar com sucesso a primeira aplicação industrial dos AGs. Desde então, estes algoritmos vêm sendo aplicados com sucesso nos mais diversos problemas de otimização e aprendizado de máquinas [9].

Um AG típico consiste em:

- uma população de possíveis soluções para o problema;
- uma maneira de avaliar o quanto cada solução é boa ou ruim para o problema;
- um método de promover a troca entre partes das soluções;
- um método para realizar mutações nas soluções com a finalidade de promover a diversidade.

2.2 Aplicações

Os AGs são eficientes na resolução de problemas complexos de otimização combinatorial. A seguir temos alguns exemplos de aplicações: [10]

- processamento de imagem;
- predição de estruturas tridimensionais de proteínas;
- *layout* de circuitos integrados VLSI (*Very Large Scale Integration*);
- tecnologia de laser;
- medicina;
- trajetórias de naves espaciais;
- roteamento de veículos terrestres;
- análise de séries temporais;
- física do estado sólido;
- aeronáutica;
- cristais líquidos;
- robótica;
- redes de água (canalização);
- desenho arquitetônico;
- evolução automática de *software*;
- escalonamento de recursos;
- reconhecimento de face;

- arquitetura e treinamento de sistemas inteligentes, como redes neurais artificiais;
- controle e automação.

2.3 Representação Cromossomial

As possíveis soluções do problema devem ser representadas por cromossomos. Os cromossomos devem ser o mais simples possível e adequados ao tipo de problema que se pretende resolver. Os operadores genéticos devem ser apropriados ao tipo de representação utilizada. Os principais tipos de cromossomos são: binários, inteiros e reais [11].

2.3.1 Representação binária

Na representação binária os indivíduos são codificados por uma sequência de dígitos binários. É uma representação geral e pode ser aplicada em qualquer problema, embora, em alguns casos, ela não seja a mais adequada.

Para exemplificar a representação binária, supõe-se que deseja-se encontrar o valor máximo da função $f(x) = |(x^2 + x)\sin(x)|$ no domínio $[-10, 10]$. Como pode-se constatar na figura 2.1, trata-se de uma função multimodal.

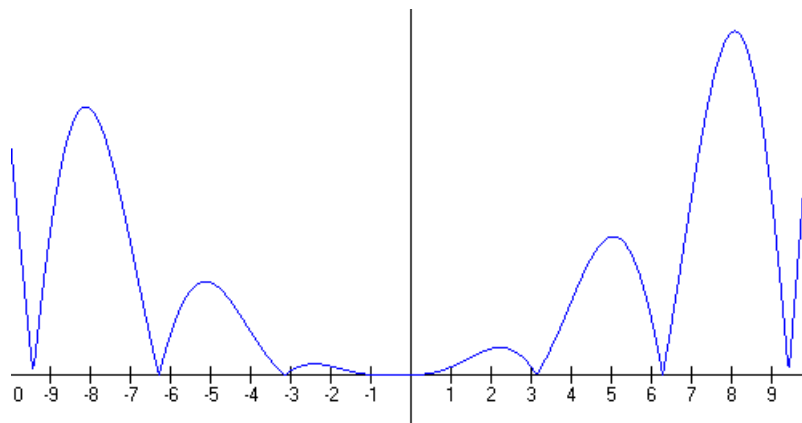


Figura 2.1: Gráfico de $f(x) = |(x^2 + x)\sin(x)|$

O problema de minimização/maximização de funções é muito simples de ser modelado com os AGs. Os indivíduos serão valores do domínio, ou seja, x . A função de aptidão

será a própria função matemática. O valor do *fitness* de um indivíduo será sua imagem, ou seja, $f(x)$. Mas, nesse caso, com a finalidade de se fazer uma representação binária dos cromossomos, é necessário quantizar os valores de x . Para isso, deve-se definir o número de bits que serão utilizados. Quanto mais bits utilizados, melhor será a precisão e maior será o número de genes do cromossomo. O número de valores que podem ser representados por um número n de bits é dado por 2^n . Assim, com 10 bits, é possível representar 1024 valores distintos. Na tabela 2.1 temos alguns exemplos de possíveis cromossomos.

Tabela 2.1: *Exemplos de cromossomos binários*

cromossomo	valor real(x)	$fitness(f(x))$
0000000000	-10.0	48.9
0100100100	-4.3	13.0
1000000000	0.0	0.0
1110101101	8.4	67.5
1111111111	10.0	59.8

2.3.2 Representação por inteiros

Na representação por inteiros os indivíduos são codificados como uma sequência de números inteiros. Geralmente é utilizada na resolução de problemas de otimização combinatorial com número finito de soluções. Ou seja, a solução do problema é uma permutação entre estados representados por números inteiros.

O problema do Caixeiro Viajante é um típico exemplo de aplicação dos cromossomos inteiros. Suponha que um caixeiro viajante deseja encontrar a melhor forma de visitar 10 cidades, passando apenas uma única vez em cada uma delas, percorrendo a menor distância possível. Atribuindo-se números de 1 a 10 para cada uma das cidades, os cromossomos então, podem ser a permutação desses 10 números, como, por exemplo a sequência abaixo:
 $C = 2\ 9\ 7\ 8\ 1\ 3\ 5\ 4\ 10\ 6$

2.3.3 Representação por reais

Na representação por reais os indivíduos são codificados como uma sequência de números reais. Nos problemas de maximização/minimização de funções pode-se também utilizar essa representação ao invés da binária.

2.4 Função de Aptidão

A função de aptidão é utilizada para determinar o grau de adaptação do indivíduo na população. Com ela é possível distinguir os bons indivíduos dentre os demais. Essa função é obtida através de uma modelagem do problema e de suas restrições. Ao submeter um indivíduo à função de aptidão obtém-se o valor de sua aptidão ou *fitness*.

No exemplo dado na subseção 2.3.1, a função de aptidão seria a própria função matemática.

2.5 Seleção

A cada iteração do AG são realizadas seleções para determinar quais indivíduos da população sofrerão a ação dos operadores genéticos. De acordo com a teoria de Darwin, somente os indivíduos mais aptos ao meio sobrevivem. Existem vários métodos de simular computacionalmente esse comportamento com os AGs.[11]

2.5.1 Método da Roleta

No Método da Roleta (*roulette-wheel*) cada indivíduo da população é representado por um setor de uma roleta circular. O tamanho de cada setor da roleta é diretamente proporcional ao valor do *fitness*. Ao girar a roleta, os melhores indivíduos terão maior probabilidade de serem selecionados.

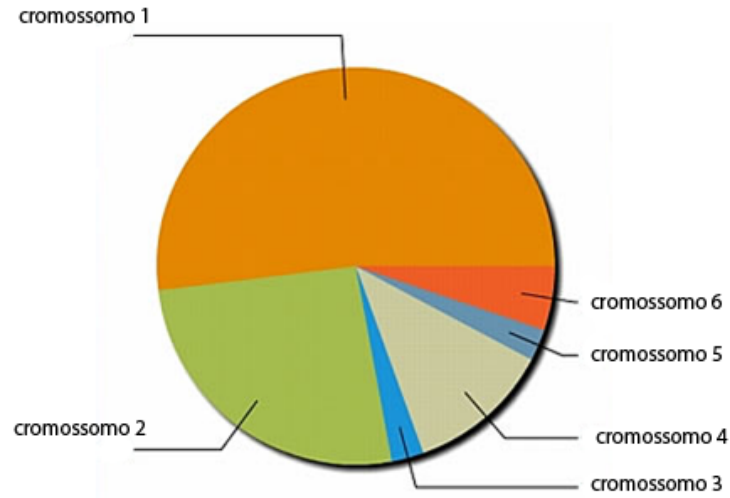


Figura 2.2: *Cromossomos na roleta*

2.5.2 Ranking

A seleção por *ranking* evita a convergência prematura e a dominância de um superindivíduo. Ela consiste em ordenar todos os indivíduos de acordo com o valor de seu *fitness* e usar esse *ranking* como base para a seleção. Pode-se utilizar o método linear para determinar a avaliação que cada indivíduo recebe:

$$E(i, t) = Min + (Max - Min) \left(\frac{rank(i, t) - 1}{N - 1} \right) \quad (2.1)$$

onde

Min é o *fitness* do pior colocado no *ranking*

Max é o *fitness* do melhor colocado no *ranking*

N é o número de indivíduos na população

$rank(i, t)$ é a posição do indivíduo i na população da geração t , classificado de acordo com o valor de seu *fitness*

2.5.3 Torneio

No método do torneio, k indivíduos são selecionados aleatoriamente, geralmente $k = 2$, então é feito um torneio entre os indivíduos selecionados. O indivíduo de maior *fitness* será o vencedor do torneio. Normalmente, retira-se o vencedor do torneio do próximo sorteio, para que não ocorra de ser selecionado duas vezes o mesmo indivíduo.

2.6 Cruzamento

O operador genético cruzamento(*crossover*) consiste na troca de material genético entre dois indivíduos gerando dois novos indivíduos. O tipo de cruzamento mais comum é o de um ponto, mas existem muitas variações desse operador de acordo com o tipo de cromossomo utilizado. Para selecionar os indivíduos que participarão do processo de mutação utiliza-se, geralmente, um dos métodos de seleção apresentados na seção 2.5, assim a probabilidade de um indivíduo da população ser selecionado para o cruzamento é proporcional ao seu *fitness*.

Os métodos apresentados nas subseções 2.6.1 e 2.6.2 são de propósito geral, porém utilizados geralmente em cromossomos binários. Já os métodos das subseções 2.6.3, 2.6.4, 2.6.5, 2.6.6 e 2.6.7 são apropriados para casos em que os genes não podem se repetir em um cromossomo, como no TSP.

2.6.1 Cruzamento de um ponto

O cruzamento de um ponto, também chamado de operador clássico de cruzamento, é geralmente aplicado a cromossomos binários. Por exemplo, sejam P_1 e P_2 , os cromossomos pais e $|$ o ponto de corte:

$$P_1 = 1\ 0\ 0\ |\ 1\ 0\ 1\ 0$$

$$P_2 = 1\ 1\ 1\ |\ 0\ 0\ 1\ 1$$

Os cromossomos resultantes serão:

$$F_1 = 1\ 0\ 0\ 0\ 0\ 1\ 1$$

$$F_2 = 1\ 1\ 1\ 1\ 0\ 1\ 0$$

2.6.2 Cruzamento de múltiplos pontos

É uma variação do cruzamento anterior, porém seleciona-se mais de um ponto de cruzamento. Por exemplo, um cruzamento de dois pontos poderia ocorrer da seguinte forma:

$$P_1 = 1\ 0\ |\ 0\ 1\ |\ 0\ 1\ 0$$

$$P_2 = 1\ 1\ |\ 1\ 0\ |\ 0\ 1\ 1$$

Os cromossomos resultantes serão:

$$F_1 = 1\ 0\ 1\ 0\ 0\ 1\ 0$$

$$F_2 = 1\ 1\ 0\ 1\ 0\ 1\ 1$$

Uma variação do cruzamento de múltiplos pontos é o Cruzamento Uniforme. Nesse cruzamento, gera-se uma máscara M de bits aleatórios. Nas posições onde a máscara é 1, o genes serão trocados nos filhos, caso contrário, serão mantidos. Por exemplo:

$$M = 1\ 1\ 0\ 0\ 1\ 1\ 0$$

$$P_1 = 1\ 0\ 0\ 1\ 0\ 1\ 0$$

$$P_2 = 1\ 1\ 1\ 0\ 0\ 1\ 1$$

Os cromossomos resultantes serão:

$$P_1 = 1\ 1\ 0\ 1\ 0\ 1\ 0$$

$$P_2 = 1\ 0\ 1\ 0\ 0\ 1\ 1$$

2.6.3 Partially-Mapped Crossover(PMX)

Para demonstrar o funcionamento do *Partially-Mapped Crossover* pode-se tomar como exemplo os cromossomos:

$$P_1 = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$$

$$P_2 = 3\ 7\ 5\ 1\ 6\ 8\ 2\ 4$$

Primeiramente seleciona-se aleatoriamente dois pontos de corte e faz-se o mapeamento dos genes entre esses pontos, como mostra a figura 2.3.

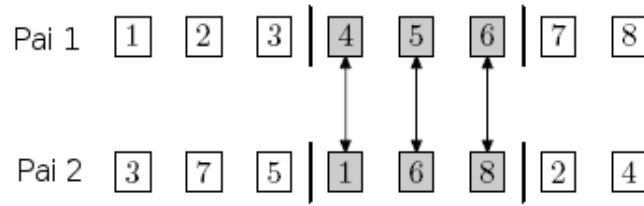


Figura 2.3: *Cromossomos pais com genes mapeados para o PMX*

Em seguida, os genes mapeados no primeiro pai são copiados para o segundo filho, e os mapeados no segundo pai serão copiados no primeiro filho, como ilustrado na figura 2.4.

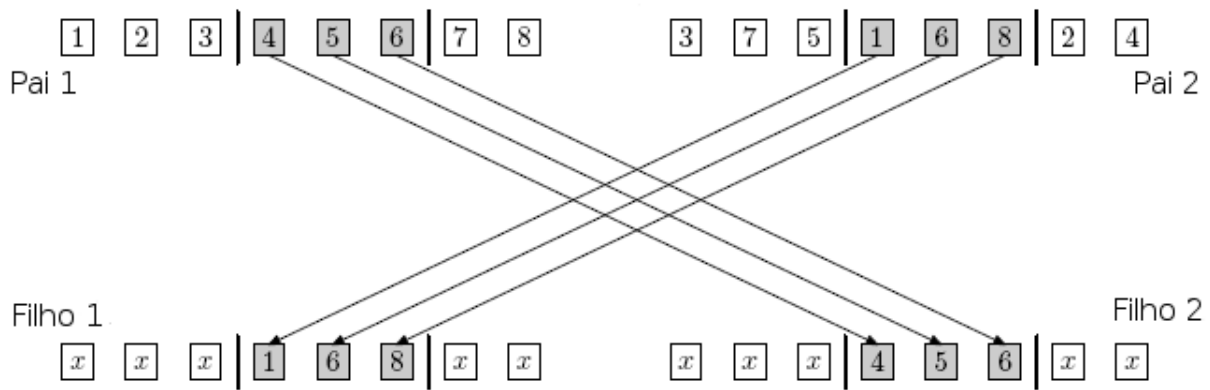


Figura 2.4: *Replicação dos genes mapeados nos filhos*

Agora os genes do primeiro pai vão sendo copiados no primeiro filho, e os genes do segundo pai vão sendo copiados para o segundo filho. Caso algum gene já esteja presente, ele é substituído por seu correspondente do mapeamento. Assim o primeiro gene do filho 1, seria 1, mas como o filho 2 já possui o gene 1 utiliza-se seu correspondente no mapeamento, ou seja, 4. O oitavo gene do filho 1 seria 8, mas o cromossomo já possui 8, ao tentar utilizar 6, seu correspondente, verifica-se que o filho 1 também já possui o gene 6. Por isso, utiliza-se o correspondente do 6, o gene 5. O filho 2 é obtido de maneira análoga. Todo esse processo pode ser melhor compreendido analisando a figura 2.5.

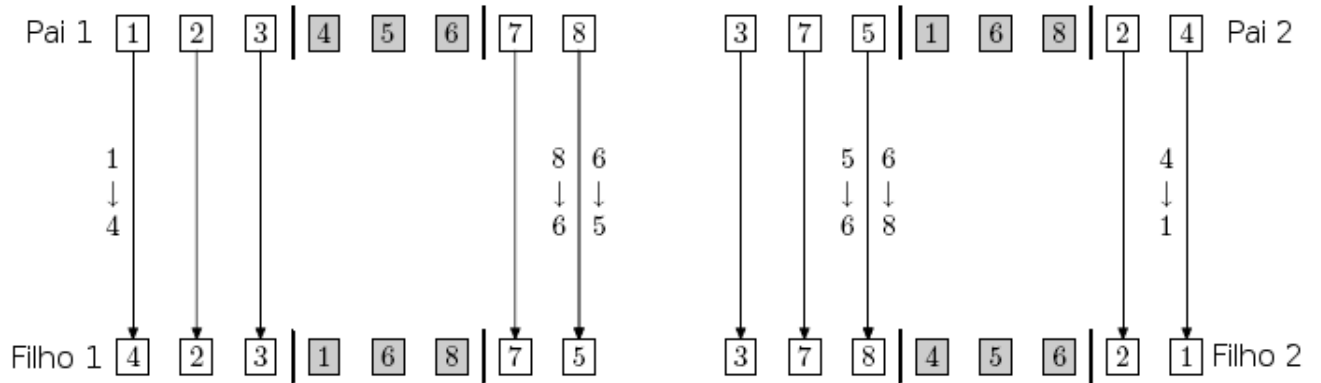


Figura 2.5: *Finalização do operador PMX*

2.6.4 Edge Recombination Crossover(ERX)

O *Edge Recombination Crossover* foi desenvolvido especialmente para ser utilizado em AGs para a resolução do TSP. No ERX, gera-se um único cromossomo contendo arestas dos pais sempre que possível. Geralmente apresenta resultados melhores que o PMX, porém possui custo computacional mais elevado.

Para exemplificar o funcionamento desse método, sejam os seguintes cromossomos:

$$P_1 = 1 \ 2 \ 6 \ 5 \ 4 \ 7 \ 3$$

$$P_2 = 7 \ 6 \ 1 \ 2 \ 3 \ 4 \ 5$$

Inicialmente cria-se uma lista de todas as arestas conectadas a cada uma das cidades. Nesse exemplo, supõe-se que $(i, j) = (j, i)$, então tem-se no mínimo duas e no máximo quatro arestas para cada cidade.

Cidade 1: (3, 1) (1, 2) (6, 1)

Cidade 2: (1, 2) (2, 6) (2, 3)

Cidade 3: (7, 3) (3, 1) (2, 3) (3, 4)

Cidade 4: (5, 4) (4, 7) (3, 4)

Cidade 5: (6, 5) (5, 4) (5, 7)

Cidade 6: (2, 6) (6, 5) (7, 6) (6, 1)

Cidade 7: (4, 7) (7, 3) (5, 7) (7, 6)

Escolhe-se a primeira cidade de um dos pais. Deve-se dar preferência para as cidades que possuam menor número de arestas. Nesse caso, tem-se:

$$F = 1 \ X \ X \ X \ X \ X \ X$$

Ao retirar das listas as arestas que contêm a cidade 1, verifica-se que a cidade que permanecerá com menor número de arestas será 2, então:

$$F = 1 \ 2 \ X \ X \ X \ X \ X$$

Retirando-se as arestas que contêm 2, as cidades 3 e 6 ficarão somente com duas arestas. Nesse caso escolhe-se aleatoriamente uma delas:

$$F = 1 \ 2 \ 6 \ X \ X \ X \ X$$

Esse processo é repetido até que o cromossomo resultante seja inteiramente preenchido. Caso a cidade de menor ligação já tenha sido incluída no cromossomo deve-se selecionar outra que ainda não tenha sido incluída. Ao término do cruzamento, tem-se:

$$F = 1 \ 2 \ 6 \ 5 \ 7 \ 3 \ 4$$

Assim tem-se um cromossomo filho gerado utilizando-se somente as arestas dos pais, sem a necessidade de gerar arestas aleatórias.

2.6.5 Cycle Crossover(CX)

No *Cycle Crossover* é identificado um número de ciclos de genes entre os pais. No primeiro ciclo, os genes pertencentes ao ciclo do pai 1 são copiados para o filho 1 e os do pai 2 para o filho 2. No segundo ciclo, os genes pertencentes ao ciclo do pai 1 são copiados para o filho 2 e os do pai 2 para o filho 1, e assim sucessivamente. Um exemplo da execução do CX está apresentado nas figuras 2.6 e 2.7.

2.6.6 Order Crossover(OX)

O operador *Order Crossover* baseia-se na ordem dos genes e não em suas posições. Os filhos são gerados utilizando-se subsequências dos genes dos pais. Sejam P_1 e P_2 os cromossomos pais e $|$ representa dois pontos de corte selecionados aleatoriamente:

$$P_1 = 1 \ 2 \ | \ 3 \ 4 \ 5 \ | \ 6 \ 7 \ 8$$

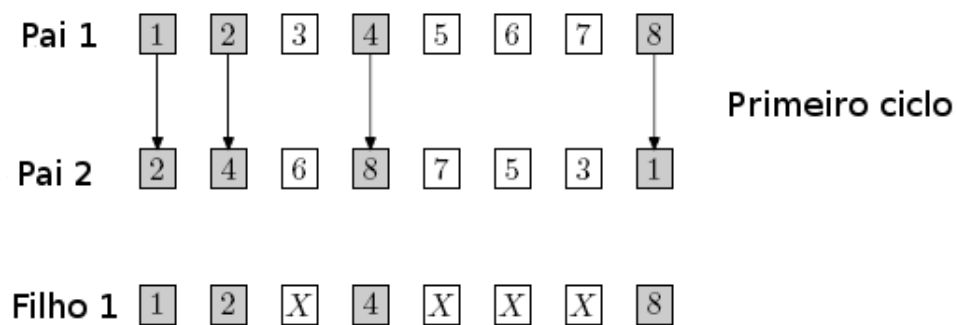


Figura 2.6: *CX - Primeiro ciclo*

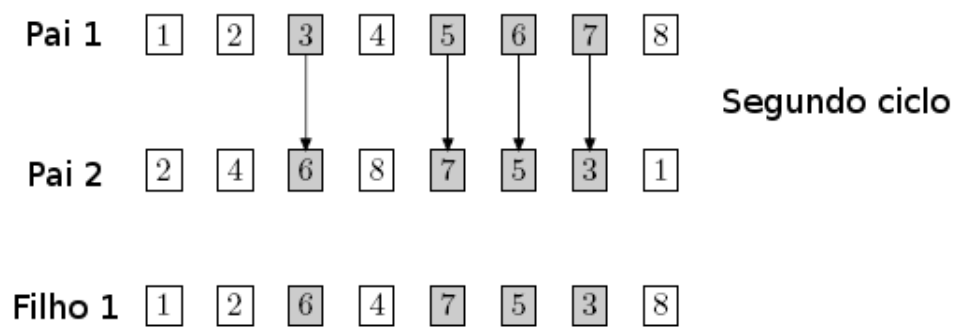


Figura 2.7: *CX - Segundo ciclo*

$$P_2 = 2\ 4\ |\ 6\ 8\ 7\ |\ 5\ 3\ 1$$

Inicialmente copiam-se os genes entre os pontos de corte diretamente para os filhos.

$$F_1 = X\ X\ |\ 3\ 4\ 5\ |\ X\ X\ X$$

$$F_2 = X\ X\ |\ 6\ 8\ 7\ |\ X\ X\ X$$

Iniciando-se do segundo ponto de corte de um dos pais, o restante dos genes são copiados a medida que aparecem no outro pai, também começando do segundo ponto de corte e omitindo os genes já presentes. Quando o fim do cromossomo é alcançado, continua-se da primeira posição. Assim:

$$F_1 = 8\ 7\ |\ 3\ 4\ 5\ |\ 1\ 2\ 6$$

$$F_2 = 4\ 5\ |\ 6\ 8\ 7\ |\ 1\ 2\ 3$$

2.6.7 Order Based Crossover(OX2)

No operador *Order Based Crossover* são selecionadas aleatoriamente várias posições de um dos pais, e essa ordem é imposta ao outro pai. Por exemplo, sejam os cromossomos:

$$P_1 = 1\ 2\ |\ 3\ 4\ 5\ |\ 6\ 7\ 8$$

$$P_2 = 2\ 4\ |\ 6\ 8\ 7\ |\ 5\ 3\ 1$$

Supondo-se que em P_2 a segunda, terceira e a sexta posição foram selecionadas. Os genes dessas posições são 4, 6 e 5, respectivamente. Em P_1 , esses genes estão presentes na quarta, quinta e sexta posições. Assim, o descendente F_1 é igual a P_1 com exceção dessas posições:

$$F_1 = 1\ 2\ 3\ X\ X\ X\ 7\ 8$$

Agora, preenchem-se as posições restantes com os genes restantes, na mesma ordem em que aparecem em P_2 :

$$F_1 = 1\ 2\ 3\ 4\ 6\ 5\ 7\ 8$$

De maneira análoga, o segundo descendente será:

$$F_2 = 2\ 4\ 3\ 8\ 7\ 5\ 6\ 1$$

2.7 Mutação

O operador de mutação tem a função de promover a variabilidade genética da população, evitando que as soluções do problema caiam em um ótimo local.

Em um cromossomo binário, o operador de mutação é realizado escolhendo-se aleatoriamente um ou mais genes e invertendo seu valor atual. Seja C , um cromossomo binário em seu estado original e C' esse cromossomo após sofrer mutação. Supondo que foi escolhido, de maneira aleatória, o quinto gene para a mutação, tem-se:

$$C = 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1$$

$$C' = 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1$$

Em cromossomos de problemas de otimização combinatorial, onde os genes não se repetem, como é o caso do TSP. Existem operadores de mutação específicos.

2.7.1 Exchange Mutation(EM)

No operador *Exchange Mutation* dois genes são selecionados aleatoriamente e trocados de posição. A EM está ilustrada na figura 2.8.

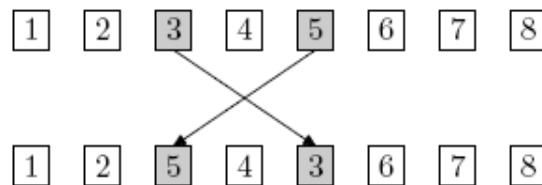


Figura 2.8: *Exemplo de EM*

2.7.2 Simple Inversion Mutation(SIM)

A *Simple Inversion Mutation* é realizada escolhendo-se aleatoriamente dois pontos de corte e invertendo-se a sequência dos genes que encontram-se entre os pontos de corte. A SIM está ilustrada na figura 2.9.

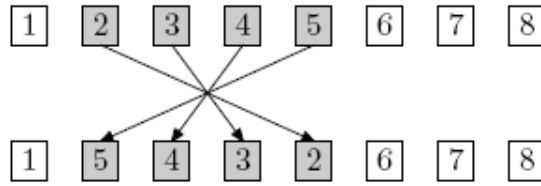


Figura 2.9: *Exemplo de SIM*

2.7.3 Insertion Mutation(ISM)

O operador *Insertion Mutation* escolhe aleatoriamente um gene no cromossomo original e o insere em uma posição aleatória, diferente da posição inicial. Exemplo na figura 2.10.

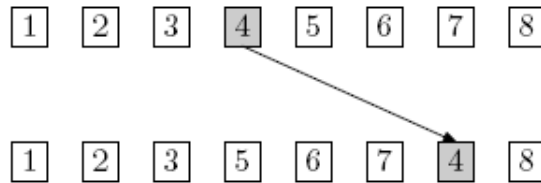


Figura 2.10: *Exemplo de ISM*

2.7.4 Inversion Mutation(IVM)

No operador *Inversion Mutation* é selecionada aleatoriamente uma subsequência de genes. Então a sequência é invertida e inserida em outra posição do cromossomo. Assim como está ilustrado na figura 2.11.

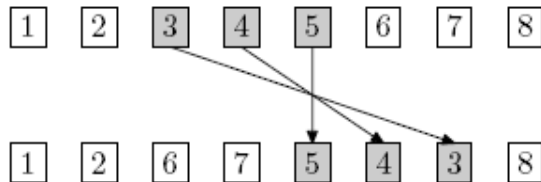


Figura 2.11: *Exemplo de IVM*

2.7.5 Scramble Mutation(SM)

No operador *Scramble Mutation*, seleciona-se aleatoriamente uma subsequência de genes e em seguida as posições desses genes são embaralhadas. Como ilustra a figura 2.12.

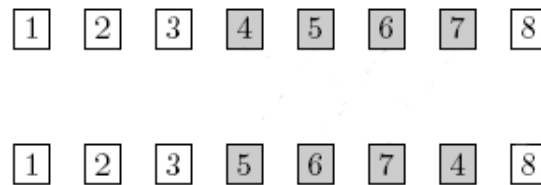


Figura 2.12: *Exemplo de SM*

2.8 Fundamentos Matemáticos

Considerando-se que um cromossomo seja representado por uma seqüência de bits. O alfabeto binário é representado por $V = \{0, 1\}$. Nesse caso, cada bit representa um gene. Assim uma representação genérica de um cromossomo de sete genes seria:

$$A = a_1 a_2 a_3 a_4 a_5 a_6 a_7$$

Em uma população de cromossomos, cada indivíduo será $A_j, j = 1, 2, \dots, n$, onde n é o número de indivíduos na população.

Um esquema (*schema*) é um padrão genético que descreve um conjunto de cromossomos do espaço de buscas com similaridades em algumas posições. O esquema será representado por H e composto pelo alfabeto $V+ = \{0, 1, *\}$. Onde o caractere $*$ representa um gene irrelevante, ou seja, pode ser 0 ou 1. Por exemplo, o cromossomo $A = 0111000$ contém o esquema $H = *11*0**$, pois os genes de H diferentes de $*$ estão presentes em A .

Seja k , o número de elementos do alfabeto de representação e l , o comprimento do cromossomo. O número de esquemas possíveis é dado por $(k+1)^l$. Como os AGs processam uma quantidade n de possíveis soluções a cada ciclo, pode-se ter noção da magnitude da informação que é processada, pois cada solução pode representar vários esquemas.

Os esquemas não são criados de maneira igual. Alguns são mais específicos que os

outros. Por exemplo, o esquema $011 * 1 * *$ possui mais similaridade que o esquema $0 * * * *$, pois o primeiro apresenta um maior número de genes definidos. Para quantificar esses padrões, foram definidas duas propriedades: a ordem do esquema(*schema order*) e o comprimento definido(*defining length*).

A ordem do esquema H , denotada por $o(H)$ é simplesmente o número de posições fixas. Por exemplo, $o(011 * 1 * *) = 4$.

O comprimento definido, denotado por $\delta(H)$, é a diferença entre a primeira e a última posição definida. Por exemplo, $\delta(011 * 1 * *) = 5 - 1 = 4$.

Os esquemas e suas propriedades são meios interessantes para a discussão e classificação das similaridades entre os cromossomos e para analisar o efeito dos operadores genéticos.

Supondo que em um intervalo de tempo t existam m exemplares de um determinado esquema H em uma população $A(t)$, onde pode-se expressar m em função do tempo como $m = m(H, t)$. Durante a reprodução, a probabilidade de um indivíduo ser selecionado é diretamente proporcional ao valor de sua aptidão(*fitness*), ou seja, para um indivíduo A_i , a probabilidade pode ser expressa como $p_i = f_i / \sum f_j$, onde f expressa o valor da aptidão. Em um instante de tempo $t + 1$ tem-se $m(H, t + 1) = m(H, t)nf(H) / \sum f_j$, onde $f(H)$ é a aptidão média dos n indivíduos que representam o esquema H no tempo t . O *fitness* médio da população pode ser escrito como $\bar{f} = \sum f_j / n$, então pode-se re-escrever a equação anterior como:

$$m(H, t + 1) \geq m(H, t) \frac{f(H)}{\bar{f}} \quad (2.2)$$

Durante o processo de cruzamento alguns indivíduos contendo o esquema H podem ser destruídos. Aplicando-se um cruzamento simples de um ponto a um cromossomo de comprimento l (genes) tem-se $l - 1$ pontos de corte. Supondo que um ponto de corte é selecionado de maneira aleatória e uniforme, a probabilidade de um esquema ser destruído é dada por $p_d = \delta(H) / (l - 1)$. Consequentemente, a probabilidade do esquema sobreviver será $p_s = 1 - p_d = 1 - \delta(H) / (l - 1)$. Além disso, ainda existe a probabilidade p_c do indivíduo ser selecionado para o cruzamento. Então a probabilidade de um esquema sobreviver ao

cruzamento é dada por:

$$p_s \geq 1 - p_c \frac{\delta(H)}{l-1} \quad (2.3)$$

Combinado-se os efeitos da reprodução e do cruzamento, o número esperado de indivíduos contendo o esquema H na próxima geração será:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{l-1} \right] \quad (2.4)$$

Finalmente, deve-se considerar o efeito do operador de mutação. Considerando uma mutação simples de um único gene, com probabilidade p_m . A probabilidade de cada gene sobreviver a mutação é de $1 - p_m$. Como o esquema H possui $o(H)$ posições fixas, a probabilidade de o esquema sobreviver a mutação será de $(1 - p_m)^{o(H)}$. Para valores pequenos de p_m , ou seja, $p_m \ll 1$, essa probabilidade pode ser simplificada para $1 - o(H)p_m$. Assim, de maneira simplificada, conclui-se que o número de elementos representantes de um determinado esquema na próxima geração será: [9]

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m \right] \quad (2.5)$$

2.9 AGs Híbridos

Em determinados problemas, os AG não são muito bons para encontrarem a solução ótima. Porém eles são muito bons para navegarem por um grande e complexo espaço de busca e encontrar soluções próximas ao ótimo global e ótimos locais. Teoricamente, um AG sempre converge para uma solução ótima, mas na prática, esse é um processo demorado. Essas limitações os tornariam menos competitivos aos processos heurísticos utilizados em aplicações reais. Principalmente na resolução de problemas de natureza combinatorial [10].

Para contornar esses problemas existem as técnicas de hibridização dos AGs. Geralmente, faz-se uma hibridização do AG com alguma heurística de busca local, levando-se em consideração as características particulares do problema que se pretende resolver. Os AGs

Híbridos apresentam várias vantagens em comparação a utilização de técnicas de heurísticas puras, porque aproveitam-se os pontos fortes de cada uma das técnicas e suprimem-se seus pontos fracos.

Capítulo 3

Implementação de um Sistema de Roteirização de Veículos

3.1 Introdução

As implementações realizadas nesse trabalho foram divididas em duas fases. Na primeira fase implementou-se uma biblioteca de classes com rotinas comuns de AGs e a partir dela, um algoritmo de roteirização. Na segunda fase, implementou-se um protótipo funcional de um sistema *web* de roteirização de veículos com interface com o usuário e banco de dados.

3.2 Tecnologias Utilizadas

3.2.1 Linguagem de Programação Java

Java é uma linguagem de programação orientada a objetos. Foi desenvolvida na década de 90 na empresa Sun Microsystems por uma equipe chefiada por James Gosling. Um programa escrito em Java é compilado para *bytecodes* e executado em uma Máquina Virtual Java (*Java Virtual Machine* - JVM), o que garante sua portabilidade para qualquer plataforma que possua uma JVM. Por ser uma linguagem de código aberto e contar com uma grande variedade de bibliotecas, *frameworks* e ferramentas de desenvolvimento gra-

tuitas, o Java se tornou rapidamente uma linguagem muito popular e é utilizada nas mais diversas áreas. A figura 3.1 mostra a diferença entre a popularidade do Java e algumas outras linguagens.[12]

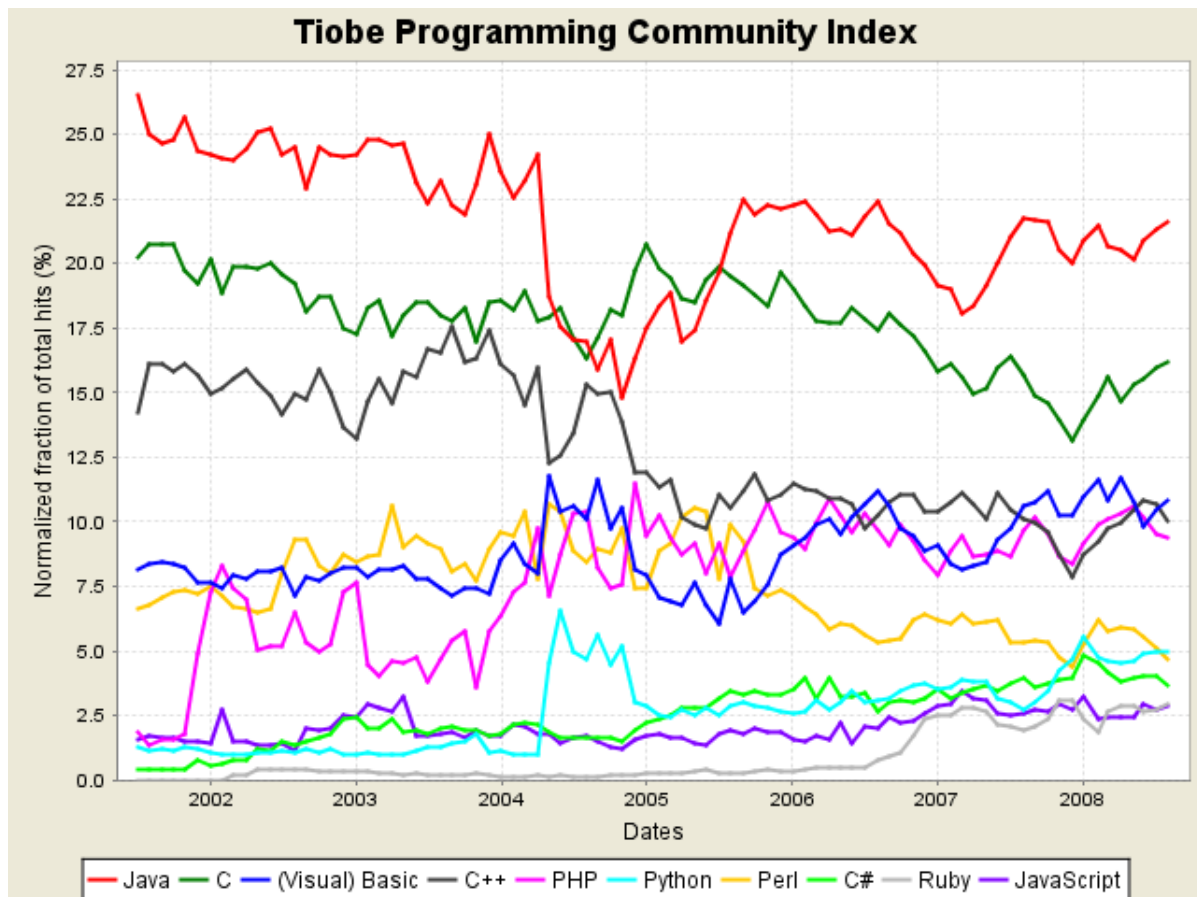


Figura 3.1: *Comparação da popularidade do Java com outras linguagens*

Entre as principais características da linguagem Java, pode-se destacar:

- Orientação a Objetos - baseado no modelo Smalltalk e Simula67;
- portabilidade, independência de plataforma;
- recursos de rede, bibliotecas que facilitam implementações envolvendo os protocolos TCP/IP, HTTP e FTP;
- sintaxe similar a linguagem C/C++;

- facilidades de internacionalização, suporta nativamente caracteres Unicode;
- facilidades para a criação de programas distribuídos e *Multi Threads*;
- coletor de lixo automático(*garbage collector*);
- carga dinâmica de código.

Durante muito tempo acreditou-se que os programas Java teriam baixo desempenho e demandariam grande quantidade de memória, mas atualmente isso não é verdade. Quando a JVM carrega o programa que será executado os *bytecodes* são compilados para código nativo pelo compilador JIT(*Just-in-Time*). A medida que o programa vai sendo executado entra em ação o *HotSpot* que faz a otimização dos processos mais utilizados na execução do programa, aumentando seu desempenho. Apesar do tempo de carga de uma aplicação Java ser um pouco maior do que de uma aplicação em código nativo, atualmente o Java apresenta desempenho próximo ao C++ em aplicações que processam grandes quantidades de dados[13, 14].

3.2.2 AJAX e Google Web Toolkit

As aplicações *web* atualmente são altamente dinâmicas e interativas graças a utilização de uma técnica denominada AJAX. Ao contrário do que muitos pensam, AJAX não é uma tecnologia, mas sim uma combinação de várias tecnologias como o JavaScript e o XML[15].

O Google Web Toolkit(GWT) é um *framework* Java, de código aberto, para a implementação de aplicações AJAX no modelo RIA(Rich Internet Applications)[16]. Nesse modelo, as aplicações *web* têm características e funcionalidades similares a aplicativos tradicionais para *desktop*. Essa vem sendo a nova tendência para aplicações *web* desde as conferências que deram início a *Web 2.0*[17].

A implementação utilizando-se diretamente HTML e JavaScript para produzir os efeitos desejáveis de uma aplicação RIA é um processo extremamente complexo e muito pouco produtivo, devido a falta de padronização entre os *browsers* existentes. O GWT elimina muitos desses problemas, pois a aplicação é codificada inteiramente em Java e as partes

referentes ao cliente são compiladas por ele para HTML e JavaScript já compatíveis com os principais *browsers* do mercado. Diferentes de tecnologias como o Adobe Flex e o Microsoft Silverlight que requerem *plugins*, o GWT não requer nenhum recurso adicional para ser executado no *browser*[18]. As partes da aplicação referentes ao servidor são compiladas normalmente para *bytecodes* Java.

O SDK(*Software Development Kit*) do GWT já possui um servidor de aplicações de um *browser* embutidos para serem utilizados na etapa de desenvolvimento da aplicação. Para ser utilizada em um ambiente de produção, deve-se fazer *deploy* da aplicação em um servidor de aplicações apropriado. Nesse trabalho, utilizou-se o Tomcat.

O Tomcat é um servidor de aplicações Java, mais especificamente um *container* de *servlets* desenvolvido pela Apache Software Foundation e distribuído como software livre [19].

3.2.3 Google Maps API

Google Maps é um serviço de visualização de mapas e imagens de satélite da Terra. É um serviço gratuito na *web* fornecido pela empresa Google. Com ele é possível traçar rotas entre quaisquer pontos da superfície terrestre. Porém, ele não faz otimização de rotas, apenas traça as rotas ligando as localidades na ordem em que o usuário vai informando.

O Google disponibiliza sua API de manipulação de mapas bem como seus serviços de busca de localidades para serem utilizados em *websites* de terceiros. Para utilizá-la, deve-se apenas fazer um cadastro e obter uma chave de utilização. Sua utilização é muito simples consistindo apenas na implementação de rotinas simples em JavaScript[20].

3.2.4 MySQL

O MySQL é um sistema de banco de dados de alto desempenho, robusto, *multi-thread* e multi-usuário. Possui código aberto e pode ser executado em diversos sistemas operacionais. Recentemente foi adquirido pela Sun Microsystems. É compatível com ODBC, JDBC e .NET, sendo capaz de interfacear com diversas linguagens de programação.

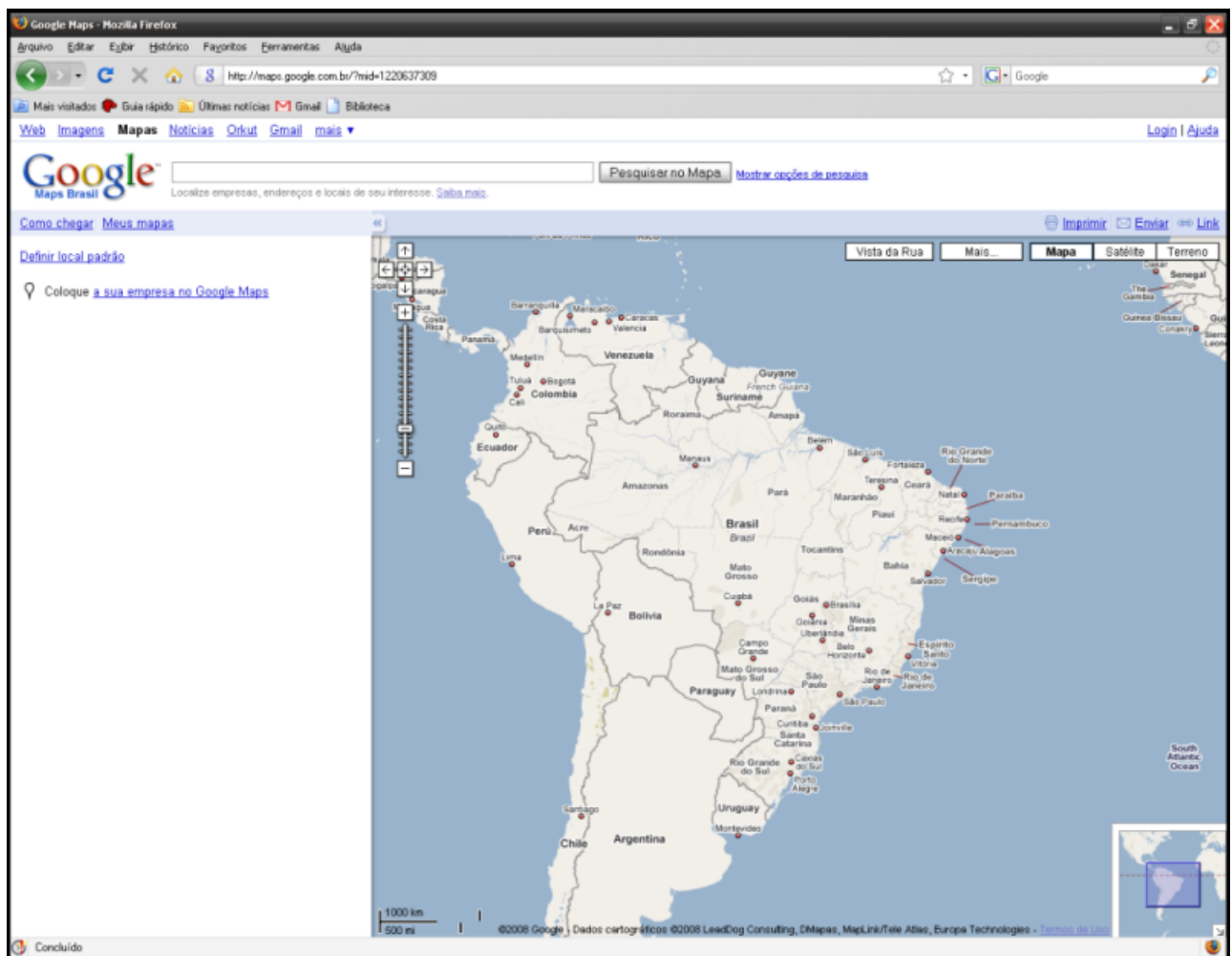


Figura 3.2: Tela inicial do Google Maps

Possui a maioria das funcionalidades presentes em grandes bancos de dados como consultas por SQL (*Structured Query Language*), controle transacional, *triggers* e *stored procedures*. Além disso, requer poucos recursos de *hardware*. [21]

3.3 Arquitetura do Sistema

Os usuários acessam a interface gráfica do sistema através do *web browser* pela internet. Nela eles são capazes de buscar e cadastrar localidades, cadastrar veículos, configurar o AG, fazer requisições de rotas e visualizar as rotas geradas.

Todo o controle e processamento do sistema fica no servidor de aplicações. A comunicação do servidor de aplicações com a interface gráfica é feita por requisições assíncronas HTTP. Dados como localidades, veículos e configurações são persistidos em um banco de dados por intermédio do servidor de aplicações.

Além da comunicação com o usuário e o banco de dados, o servidor de aplicações também comunica-se com os servidores do Google para obter os mapas e informações de trajetos entre as localidades. O diagrama da figura 3.3 mostra como o sistema está distribuído.

3.4 Banco de Dados

Algumas informações tratadas pelo sistema são armazenadas em um banco de dados MySQL. Normalmente esse banco encontra-se na mesma máquina que o servidor de aplicações ou em uma outra máquina na rede local, fazendo que a comunicação entre eles seja bem eficiente.

Os dados referentes a cada uma das localidades que é cadastrada pelo usuário são salvos na tabela de Localidades(TLOC). Essa tabela possui os seguintes campos:

- CODIGO: número inteiro utilizado como identificador único para cada localidade. Esse campo é também utilizado na montagem dos cromossomos como representação das localidades.

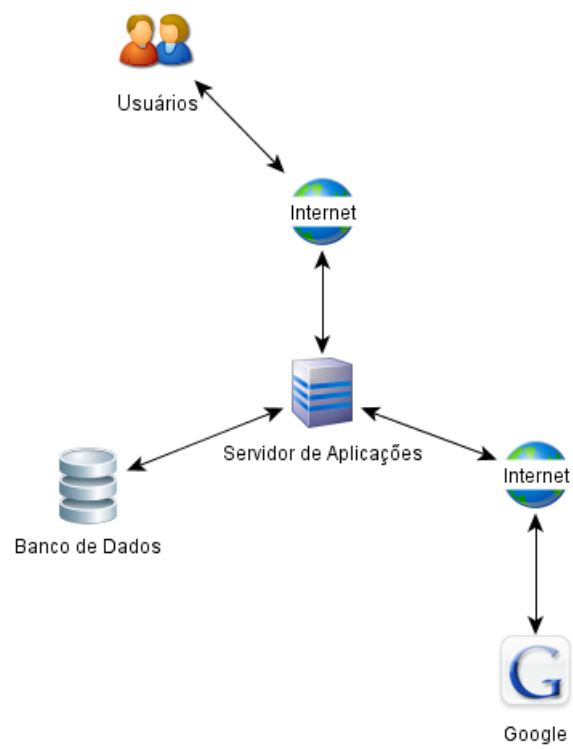


Figura 3.3: *Arquitetura do sistema*

- DESCRICAO: breve descrição sobre a localidade.
- LATITUDE: valor da latitude da localidade.
- LONGITUDE: valor da longitude da localidade.
- ENDERECO: endereço da localidade.

Na tabela de Distâncias (TDIST) são armazenados os valores de distâncias entre as localidades duas a duas. Como o sistema considera os sentidos das vias, normalmente a distância entre as localidades A e B é diferente da distância entre B e A, os campos dessa tabela são:

- CODORIGEM: identificador único da localidade de origem.
- CODDESTINO: identificador único da localidade de destino.
- DISTÂNCIA: distância, em quilômetros, entre as duas localidades. Essa distância não é a distância em linha reta entre as localidades, ela já considera o trajeto pelas vias de circulação.

Como o sistema é capaz de processar múltiplas rotas, deve-se conhecer os veículos disponíveis para serem utilizados. Esse dados são armazenados na tabela de veículos (TVEIC), cujos campos são:

- CODIGO: número inteiro utilizado como identificador único de cada veículo.
- DESCRICAO: breve descrição sobre o veículo.
- CAPACIDADE: capacidade do veículo.

O parâmetros do AG são configuráveis pelo usuário e são salvos na tabela de Parâmetros (TPAR), que possui os seguintes campos:

- NOME: nome do parâmetro

- TIPO: tipo do parâmetro. Para esse campo, convencionou-se 'I' para parâmetros do tipo inteiro, 'D' para parâmetros do tipo decimal e 'T' para valores do tipo texto.
- VALOR: valor do parâmetro salvo como texto.

Na figura 3.4 tem-se o Diagrama de Entidade-Relacionamento(DER) do banco de dados utilizado pelo sistema.

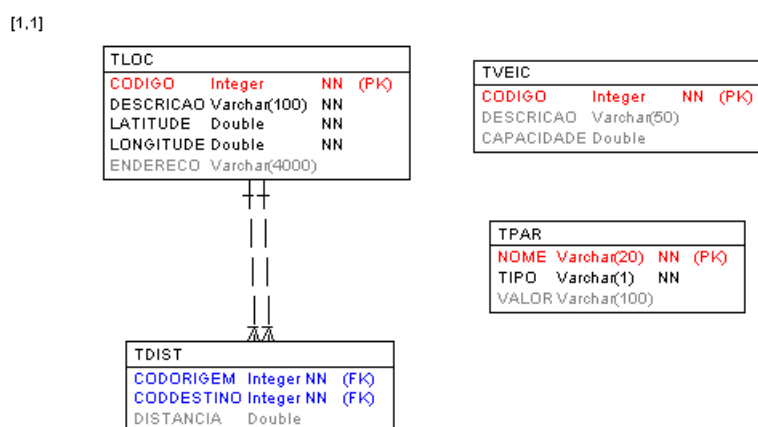


Figura 3.4: Diagrama de Entidade-Relacionamento do banco de dados

3.5 Implementação do AG

Nesse trabalho não implementou-se apenas um AG específico para a resolução do problema em questão e sim uma biblioteca ampla que pode ser reutilizada para a resolução de outros problemas. Foram implementados diversos tipos de cromossomos, operadores genéticos, tipos de seleção etc.

A biblioteca foi desenvolvida na linguagem de programação Java, utilizando-se o JDK 1.6.0_05. A implementação foi feita utilizando-se o paradigma de orientação a objetos tornando a biblioteca portátil e extensível.

O algoritmo de roteirização é a parte inteligente do sistema, não possui nenhuma dependência ou ligação direta com a interface gráfica e o banco de dados. Trata-se de um

Algoritmo Genético Híbrido implementado utilizando-se partes da biblioteca de AGs para a resolução do TSP.

Foram utilizados cromossomos do tipo inteiro, onde cada gene é um número inteiro único representando uma localidade. O algoritmo oferece suporte a múltiplas rotas a partir um ponto central, que poderia ser, por exemplo, um centro de distribuição de mercadorias.

Para que fosse possível a geração de múltiplas rotas, foi necessário fazer uma adaptação na estrutura dos cromossomos. O centro de distribuições foi inserido no cromossomo n vezes, onde n é o número de rotas desejáveis, porém com identificadores diferentes, ou seja, o AG trata como pontos distintos, mas na realidade é o mesmo ponto inserido mais de uma vez.

A inicialização do cromossomo é feita com sequências válidas aleatórias, representando possíveis rotas.

O algoritmo permite a utilização de três métodos de cálculo para o custo de traslado entre os pontos. O primeiro considera que os pontos estão localizados sobre um plano cartesiano. Assim, para os pontos $P_1(x_1, y_1)$ e $P_2(x_2, y_2)$, a distância entre eles é dada pela equação 3.1

$$d_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (3.1)$$

O segundo considera o Sistema de Coordenadas Geográficas. Um ponto pode ser localizado na superfície da Terra de acordo com o valor de duas coordenadas, a latitude e a longitude. A latitude é a distância ao Equador, em graus, ao longo do meridiano de Greenwich. Essa distância pode variar de 0 a 90° para Norte ou para Sul. A longitude é a distância, em graus, ao meridiano de Greenwich medida ao longo do Equador. Pode variar de 0 a 180° para Leste ou Oeste. Na figura 3.5, a latitude é o valor do ângulo φ e a longitude o ângulo λ .

Assim, a distância entre os pontos $P_1(\varphi_1, \lambda_1)$ e $P_2(\varphi_2, \lambda_2)$, é calculada pela equação 3.2.

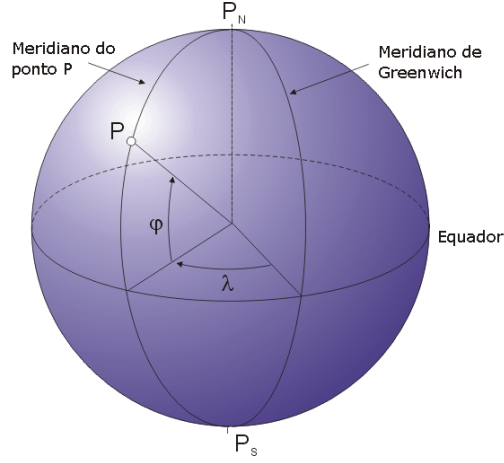


Figura 3.5: *Coordenadas geográficas na esfera*

$$d_{12} = \arccos(\cos(\lambda_2 - \lambda_1)\sin(90^\circ - \varphi_2)\sin(90^\circ - \varphi_1) + \cos(90^\circ - \varphi_2)\cos(90^\circ - \varphi_1)) \quad (3.2)$$

O terceiro método não faz cálculos de distâncias. Ele utiliza uma matriz de custos do traslado entre as cidades. Com a utilização desse método podem-se considerar outros fatores além da distância para a determinação do custo, deixando o algoritmo bastante flexível.

O operador de cruzamento utilizado foi o PMX, utilizando-se o método da roleta para fazer a seleção. O operador de mutação utilizado foi a *Exchange Mutation*. Sendo tanto a taxa de mutação quanto a taxa de cruzamento configuráveis.

É possível utilizar a substituição direta ou o elitismo no AG. No caso da utilização do elitismo, a quantidade de indivíduos que sobreviverão é configurável.

O AG foi hibridizado com a técnica de busca local 2opt [22], melhorando assim sua capacidade e seu desempenho. Essa técnica é particular para a resolução do TSP, pois uma determinada parcela das possíveis rotas são analisadas e melhoradas através da inversão de sub-rotas. No exemplo da figura 3.6, os segmentos $\overline{ab}$ e $\overline{cd}$ visivelmente não são uma boa sub-rota. Para um humano isso é fácil de ser concluído, mas o AG puro tem dificuldade de tomar esse tipo de decisão. Assim o operador 2opt substituiria essas rotas por $\overline{ac}$ e $\overline{bd}$,

pois $\overline{ab} + \overline{cd} > \overline{ac} + \overline{bd}$.

O operador 2opt percorre o cromossomo tomando pares seqüentes de genes e compara a distância percorrida total da rota original com a da rota com os dois genes invertidos. Se a nova distância percorrida for maior ou igual a original, o cromossomo original é mantido e avaliam-se os próximos pares. Caso a nova distância seja menor, o cromossomo é alterado e a avaliação recomeça. O cromossomo é analisado até que não seja mais possível fazer mais melhorias na rota.

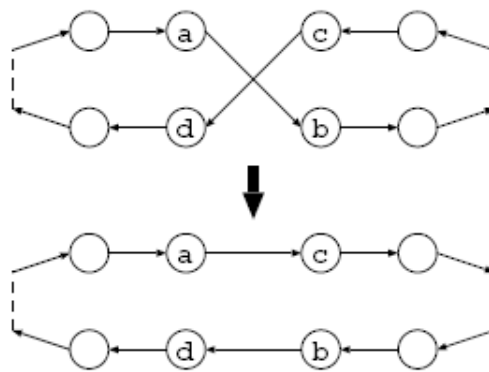


Figura 3.6: *Exemplo do operador 2opt*

3.6 Implementação da Aplicação *web*

Optou-se por implementar o sistema de roteirização como uma aplicação *web* ao invés de um aplicativo *desktop*, devido a diversos motivos:

- Atualmente a *web* com todos seus recursos consolidou-se como uma plataforma de desenvolvimento de aplicações e serviços e não mais simplesmente como uma provedora de conteúdo e páginas estáticas.
- Multi-usuários. Cada usuário deve ser capaz de acessá-la mesmo estando fora de seu local de trabalho e sem a necessidade de instalar nenhum recurso adicional, tendo em vista que os principais sistemas operacionais já trazem por padrão pelo menos um *browser* instalado.

- Multi-plataforma. Ela deve ser capaz de ser executada em qualquer plataforma que o usuário esteja trabalhando.
- Facilidade da utilização do Google Maps API. Com a API de mapas do Google é muito fácil embutir um mapa até mesmo em um simples *website*.
- A parte inteligente do sistema fica centralizada em um servidor, que geralmente é uma máquina bem mais potente que uma estação de trabalho.

A aplicação *web* foi implementada utilizando-se o GWT 1.5.2 e JDK 1.6.0_05. Ela é capaz de traçar rotas reais sobre o mapa considerando rodovias, avenidas e ruas, inclusive o sentido de circulação do tráfego.

Para a utilização da aplicação, primeiramente é necessário que sejam cadastradas as localidades por onde serão traçadas as rotas. As localidades podem ser qualquer ponto na superfície terrestre, como por exemplo, pontos de uma cidade, cidades em um país etc. São cadastradas as informações mostradas na figura 3.7, mas somente o valor da latitude e longitude são utilizadas para a roteirização.

Id	Nome	Latitude	Longitude
1	Ag. Central	-18.91749	-48.281007
2	Ag. Roosevelt	-18.91749	-48.281007

Figura 3.7: *Cadastro de localidade*

Caso as coordenadas da localidade que está sendo cadastrada não sejam conhecidas, a aplicação consegue determiná-las bastando para isso fornecer alguma informação como endereço, nome da cidade, nome do país etc, como está exemplificado na figura 3.8. Além disso, ao fazer uma busca da localidade, ela é automaticamente indicada no mapa.

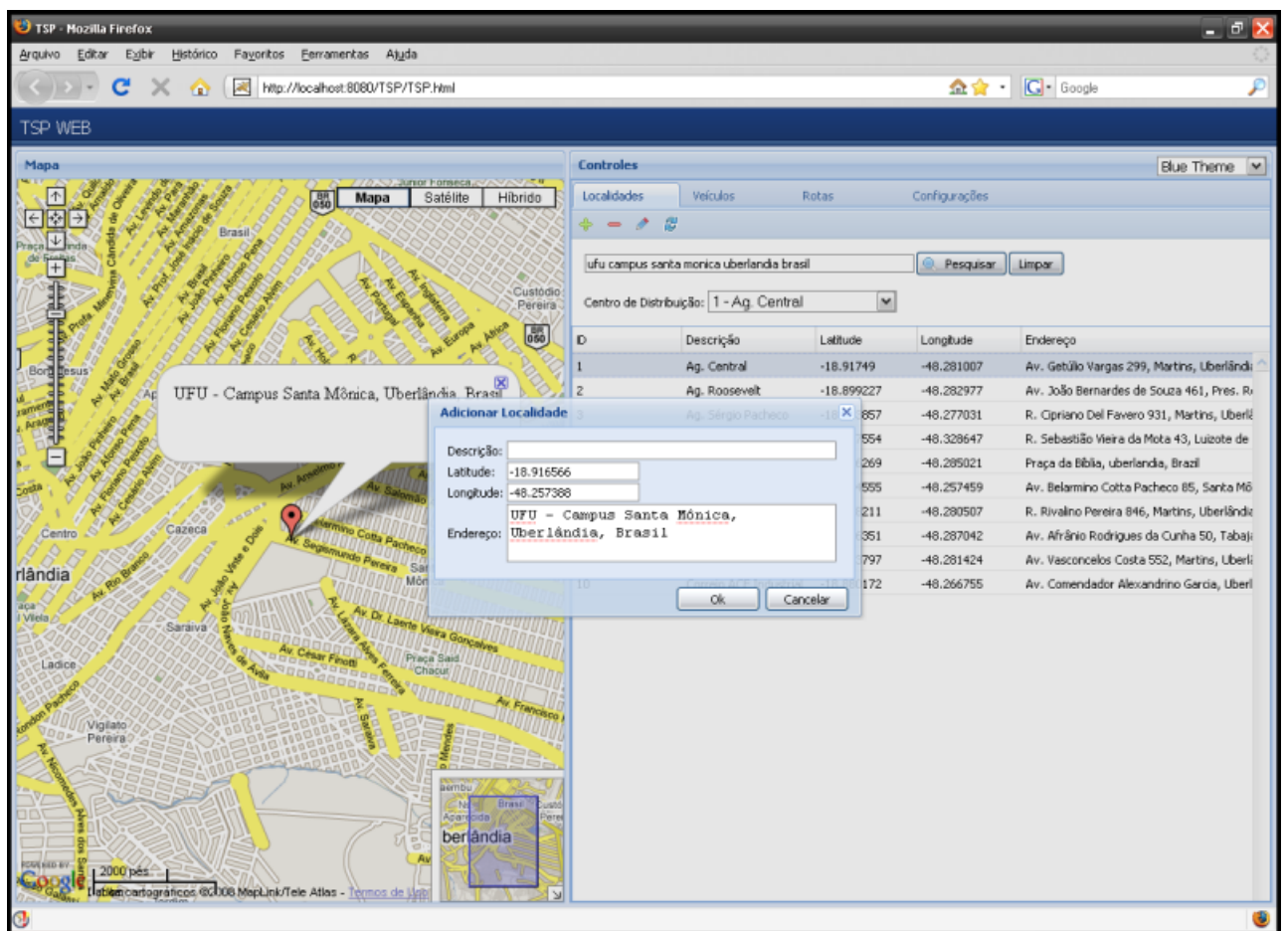


Figura 3.8: Exemplo de busca das coordenadas de localidade

Para a utilização de múltiplas rotas, deve-se selecionar uma das localidades cadastradas para ser o centro de distribuição.

Deve-se cadastrar os veículos que estarão disponíveis para percorrer as rotas. Porém, ao gerar uma rota, pode ocorrer de nem todos os veículos serem utilizados, pois o algoritmo decide qual o número de veículos utilizados será mais vantajoso, considerando-se critérios como distância percorrida e tempo.

Os parâmetros do AG são configuráveis diretamente pela aplicação *web* na aba “Configurações”. São eles:

- número de indivíduos;
- número de gerações;
- probabilidade de cruzamento;
- probabilidade de mutação;
- taxa de elitismo.

Após cadastradas as localidades e os veículos e configurados os parâmetros do AG, o sistema está pronto para fazer a geração das rotas. Para isso, basta selecionar a aba “Rotas” e clicar no botão “Executar”. Será enviada uma requisição de roteirização ao servidor. O servidor fará consultas no banco de dados para obter as informações necessárias a execução do algoritmo. Ele fará também requisições aos serviços do Google Maps para obter informações de trajetórias. De posse dos dados necessários, ele fará a execução do AG e retornará as informações da trajetória encontrada visualmente, traçada sobre o mapa e também uma lista, descrevendo detalhadamente o trajeto.

O mapa é completamente interativo, permitindo ao usuário que o movimente livremente e aplique mais ou menos *zoom* de acordo com sua necessidade. As localidades que fazem parte da rota são apontadas por marcadores. Ao clicar sobre o marcador, serão exibidas informações sobre a localidade. A rota a ser percorrida pelo veículo é traçada sobre o mapa exatamente como deve ser percorrida, considerando o sentido de circulação das vias.

Além do mapa será exibida uma lista com as localidades, duas a duas, na ordem em que deverão ser visitadas. Para cada item da lista, será apresentada a distância entre as localidades. Essa distância não é a distância linear, e sim a distância real que será percorrida, já levando-se em conta as vias de circulação. É apresentado também o tempo aproximado que o veículo levará para percorrer cada parte do trajeto. Esse tempo é estimado considerando-se a velocidade máxima de circulação de acordo com o tipo de via (rua, avenida, rodovia).

Um exemplo de como as rotas são apresentas pelo sistema, pode ser visto na figura 3.9.

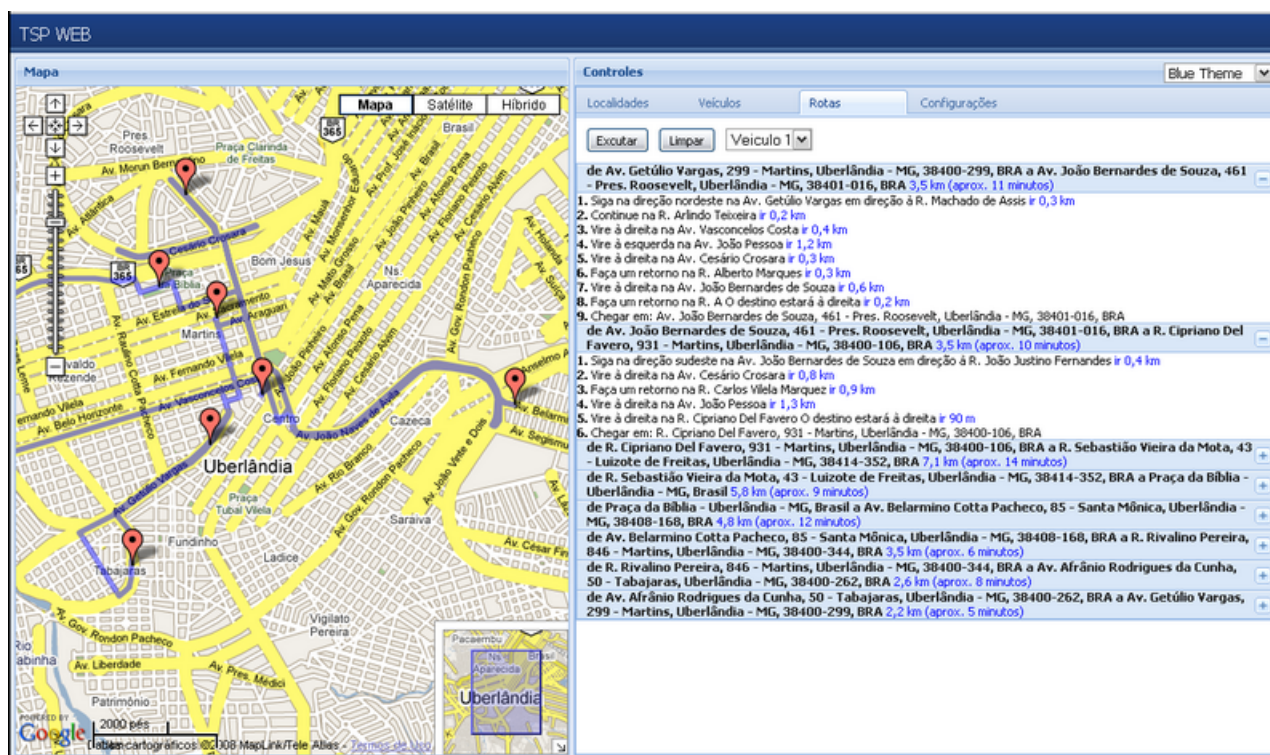


Figura 3.9: Exemplo de rota gerada

Capítulo 4

Testes e Resultados

4.1 Introdução

Após a implementação ter sido concluída foram realizados vários testes com a finalidade de comprovar a eficiência do AG implementado. Foram realizados também testes com a interface gráfica com o objetivo de verificar suas funcionalidades e seu grau de facilidade de uso. Realizou-se testes variando-se o número e disposição de localidades, número de veículos disponíveis e parâmetros do AG.

Neste capítulo, serão mostrados dois testes realizados. O primeiro tem como objetivo comprovar o desempenho do AG puro, calculando-se uma rota sobre o plano cartesiano. No segundo, pretende-se demonstrar a eficiência do sistema como um todo utilizando-se como exemplo um problema real.

4.2 Teste do AG

Nesse teste foram dispostos 20 pontos em um plano cartesiano, cujas coordenadas estão na tabela 4.1. Para o cálculo do custo do deslocamento entre dois pontos utilizou-se a distância euclidiana.

O AG foi executado com os seguintes parâmetros:

Tabela 4.1: *Coordenadas no plano cartesiano*

Ponto	x	y
A	290	20
B	213	25
C	153	48
D	93	74
E	62	128
F	58	219
G	117	289
H	180	319
I	259	344
J	336	280
K	413	305
L	470	272
M	496	212
N	341	166
O	457	121
P	445	91
Q	402	50
R	298	97
S	367	36
T	333	27

- Número de indivíduos: 20
- Gerações: 50
- Probabilidade de cruzamento: 60%
- Probabilidade de mutação: 5%
- Taxa de elitismo: 10%

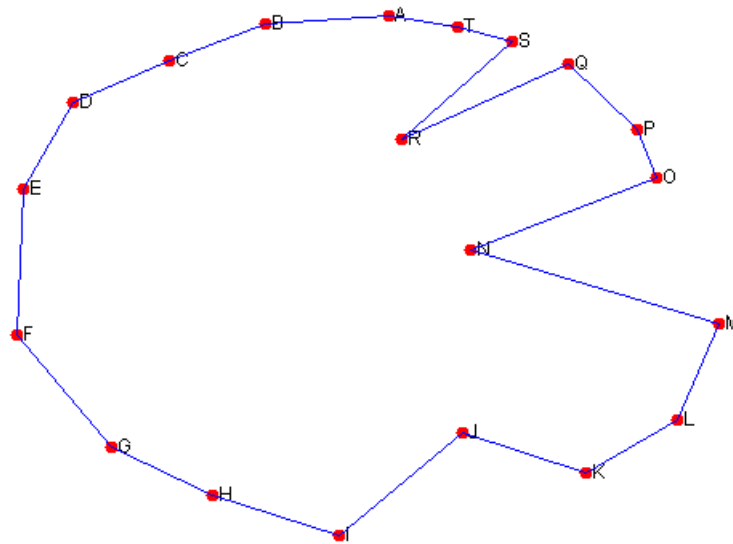


Figura 4.1: *Rota gerada pelo AG*

Analisando-se a figura 4.1 pode-se comprovar que o AG encontrou uma boa solução, pois como a maioria dos pontos estão dispostos em uma configuração aproximadamente circular era de se esperar que a rota ótima percorresse a circunferência. Além disso, mesmo com a presença de pontos que fogem ao padrão de uma circunferência, nota-se que não obteve-se trajetórias se cruzando. Apesar de terem sido executadas 50 gerações, o AG convergiu logo após a 26^a geração, como pode ser comprovado pelo gráfico 4.2 que ilustra a evolução da população ao longo das gerações.

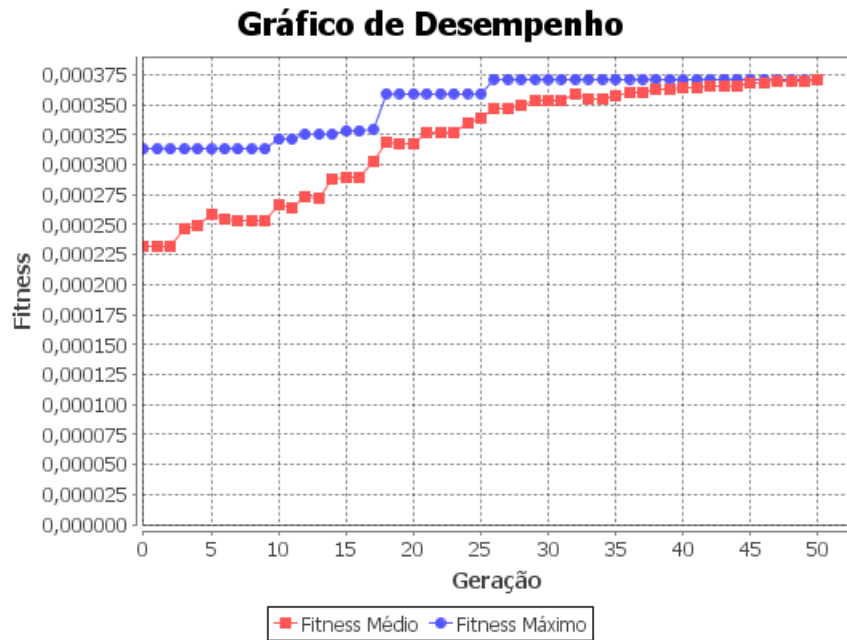


Figura 4.2: *Gráfico de Desempenho do AG*

4.3 Teste do Sistema

Para demonstrar como um problema complexo pode ser facilmente resolvido pelo sistema desenvolvido e a forma como as rotas são apresentadas, tomou-se como exemplo o seguinte problema:

“Sejam as 10 agências dos Correios da cidade de Uberlândia listadas na tabela 4.2. Supondo que as cartas e encomendas que sejam postadas em cada uma delas devam ser enviadas até a agência central. Supondo que existam 2 furgões que partirão da agência central, farão a coleta e retornarão ao ponto de partida, qual seria a melhor rota a ser percorrida por cada um dos veículos?”

Aparentemente parece um problema simples, uma vez que, pensando no problema do TSP, tem-se apenas 10 pontos para serem percorridos. Mas esse problema é bem mais complexo do que parece, pois como os pontos correspondem a localidades em uma cidade que serão visitados por veículos, deve-se considerar as regras de circulação no trânsito, então, dificilmente a distância entre dois pontos quaisquer A e B será a mesma que a distância entre B e A. Além disso, provavelmente existirão 2 sub-rotas, aumentando ainda

Tabela 4.2: *Agências dos Correios*

Agência	Latitude	Longitude	Endereço
Ag. Central	-18,917449	-48,281007	Av. Getúlio Vargas, 299
Ag. Roosevelt	-18,899227	-48,282977	Av. João Bernardes de Souza, 461
Ag. Sérgio Pacheco	-18,899227	-48,277031	R. Cipriano Del Favero, 931
Ag. Luizote de Freitas	-18,917554	-48,328647	R. Sebastião Vieira da Mota, 43
Correio Rodoviária	-18,906269	-48,285021	Praça da Bíblia
Correio Santa Mônica	-18,914555	-48,257459	Av. Belarmino Cotta Pacheco, 85
Ag. Martins	-18,908211	-48,280507	R. Rivalino Pereira, 846
Correios Tabajaras	-18,926351	-48,287042	Av. Afrânio Rodrigues da Cunha, 50
Correios Vila Osvaldo	-18,913797	-48,281424	Av. Vasconcelos Costa, 552
Correio ACF Industrial	-18,880172	-48,266755	Av. Comendador Alexandrino Garcia

mais o número de soluções possíveis.

O AG foi executado com os seguintes parâmetros:

- Número de indivíduos: 100
- Gerações: 100
- Probabilidade de cruzamento: 65%
- Probabilidade de mutação: 5%
- Taxa de elitismo: 10%

Como resultado obteve-se para o veículo 1, a rota apresentada na figura 4.3, e para o veículo 2, a rota da figura 4.4. O sistema também apresentou a descrição detalhada da rota apresentada a seguir.

O veículo 1 deve percorrer a seguinte rota:

de Av. Getúlio Vargas, 299 - Martins, Uberlândia - MG, 38400-299, BRA a Av. João Bernardes de Souza, 461 - Pres. Roosevelt, Uberlândia - MG, 38401-016, BR 3,5 km (aprox. 11 minutos)

1. Siga na direção nordeste na Av. Getúlio Vargas em direção à R. Machado de Assis
ir 0,3 km

2. Continue na R. Arlindo Teixeira ir 0,2 km
3. Vire à direita na Av. Vasconcelos Costa ir 0,4 km
4. Vire à esquerda na Av. João Pessoa ir 1,2 km
5. Vire à direita na Av. Cesário Crosara ir 0,3 km
6. Faça um retorno na R. Alberto Marques ir 0,3 km
7. Vire à direita na Av. João Bernardes de Souza ir 0,6 km
8. Faça um retorno na R. A O destino estará à direita ir 0,2 km
9. Chegar em: Av. João Bernardes de Souza, 461 - Pres. Roosevelt, Uberlândia - MG, 38401-016, BRA

de Av. João Bernardes de Souza, 461 - Pres. Roosevelt, Uberlândia - MG, 38401-016, BRA a R. Cipriano Del Favero, 931 - Martins, Uberlândia - MG, 38400-106, BRA 3,5 km (aprox. 10 minutos)

1. Siga na direção sudeste na Av. João Bernardes de Souza em direção à R. João Justino Fernandes ir 0,4 km
2. Vire à direita na Av. Cesário Crosara ir 0,8 km
3. Faça um retorno na R. Carlos Vilela Marquez ir 0,9 km
4. Vire à direita na Av. João Pessoa ir 1,3 km
5. Vire à direita na R. Cipriano Del Favero O destino estará à direita ir 90 m
6. Chegar em: R. Cipriano Del Favero, 931 - Martins, Uberlândia - MG, 38400-106, BRA

de R. Cipriano Del Favero, 931 - Martins, Uberlândia - MG, 38400-106, BRA a R. Sebastião Vieira da Mota, 43 - Luizote de Freitas, Uberlândia - MG, 38414-352, BRA 7,1 km (aprox. 14 minutos)

1. Siga na direção nordeste na R. Cipriano Del Favero em direção à R. Ricardo Felice ir 41 m
2. Vire à esquerda na R. Ricardo Felice ir 0,1 km
3. Vire à esquerda na Av. Vasconcelos Costa ir 2,7 km
4. Vire à direita na Av. Geraldo Mota Batista ir 0,3 km
5. Vire à esquerda na Av. Brg. Sampaio ir 0,5 km
6. Vire à direita na Av. Asp. Mega ir 1,8 km
7. Na rotatória, pegue a 3ª saída para a Av. José Fonseca E Silva ir 1,5 km
8. Vire à esquerda na R. Genarino Cazabono ir 0,1 km
9. Curva acentuada à direita na R. Sebastião Vieira da Mota O destino estará à esquerda ir 40 m
10. Chegar em: R. Sebastião Vieira da Mota, 43 - Luizote de Freitas, Uberlândia - MG, 38414-352, BRA

de R. Sebastião Vieira da Mota, 43 - Luizote de Freitas, Uberlândia - MG, 38414-352, BRA a Praça da Bíblia - Uberlândia - MG, Brasil 5,8 km (aprox. 9 minutos)

1. Siga na direção oeste na R. Sebastião Vieira da Mota em direção à Av. José Fonseca E Silva ir 0,1 km
2. Vire à direita na Av. José Fonseca E Silva Passe por 1 rotatória ir 1,9 km
3. Vire à direita na Av. Rubi ir 0,4 km
4. Vire à esquerda na Av. Platina ir 0,2 km
5. Vire à direita para pegar a Av. Juscelino Kubitscheck ir 0,9 km
6. Curva suave à direita na BR-365 ir 1,3 km

7. Continue na Av. Profa. Minervina Cândida de Oliveira ir 0,9 km
8. Vire à direita na R. Rodrigues da Cunha ir 0,1 km
9. Vire à esquerda na R. Indianópolis ir 0,2 km
10. Chegar em: Praça da Bíblia - Uberlândia - MG, Brasil

de Praça da Bíblia - Uberlândia - MG, Brasil a Av. Belarmino Cotta Pacheco, 85 - Santa Mônica, Uberlândia - MG, 38408-168, BRA 4,8 km (aprox. 12 minutos)

1. Siga na direção leste na R. Indianópolis em direção à R. Vieira Gonçalves ir 0,1 km
2. Vire à direita na R. Vieira Gonçalves ir 0,2 km
3. Vire à esquerda na Av. Estrela do Sul ir 0,5 km
4. Vire à direita na Av. João Pessoa ir 1,1 km
5. Vire à esquerda na Av. Afonso Pena ir 0,1 km
6. Vire à direita em direção à Av. João Naves de Ávila ir 0,2 km
7. Curva suave à esquerda na Av. João Naves de Ávila ir 2,1 km
8. Na rotatória, pegue a 4ª saída para a rampa de acesso à Av. João Naves de Ávila ir 0,1 km
9. Vire à esquerda na Av. João Naves de Ávila ir 0,2 km
10. Sair na Av. Belarmino Cotta Pacheco O destino estará à esquerda ir 0,1 km
11. Chegar em: Av. Belarmino Cotta Pacheco, 85 - Santa Mônica, Uberlândia - MG, 38408-168, BRA

de Av. Belarmino Cotta Pacheco, 85 - Santa Mônica, Uberlândia - MG, 38408-168, BRA a Av. Getúlio Vargas, 299 - Martins, Uberlândia - MG, 38400-299, BRA 3,6 km (aprox. 6 minutos)

1. Siga na direção oeste na Av. Belarmino Cotta Pacheco ir 78 m
2. Pegue a rampa de acesso a Av. João Naves de Ávila ir 55 m
3. Vire à direita na Av. João Naves de Ávila ir 2,1 km
4. Continue na Av. Belo Horizonte ir 0,3 km
5. Vire à esquerda na Av. João Pinheiro ir 0,8 km
6. Vire à direita na R. Duque de Caxias ir 0,2 km
7. Vire à direita na Av. Getúlio Vargas O destino estará à direita ir 65 m
8. Chegar em: Av. Getúlio Vargas, 299 - Martins, Uberlândia - MG, 38400-299, BRA

O veículo 2 deve percorrer a rota a seguir:

de Av. Getúlio Vargas, 299 - Martins, Uberlândia - MG, 38400-299, BRA a Av. João Bernardes de Souza, 461 - Pres. Roosevelt, Uberlândia - MG, 38401-016, BRA 3,5 km (aprox. 11 minutos)

1. Siga na direção nordeste na Av. Getúlio Vargas em direção à R. Machado de Assis ir 0,3 km
2. Continue na R. Arlindo Teixeira ir 0,2 km
3. Vire à direita na Av. Vasconcelos Costa ir 0,4 km
4. Vire à esquerda na Av. João Pessoa ir 1,2 km
5. Vire à direita na Av. Cesário Crosara ir 0,3 km
6. Faça um retorno na R. Alberto Marques ir 0,3 km
7. Vire à direita na Av. João Bernardes de Souza ir 0,6 km
8. Faça um retorno na R. A O destino estará à direita ir 0,2 km

9. Chegar em: Av. João Bernardes de Souza, 461 - Pres. Roosevelt, Uberlândia - MG, 38401-016, BRA

de Av. João Bernardes de Souza, 461 - Pres. Roosevelt, Uberlândia - MG, 38401-016, BRA a R. Cipriano Del Favero, 931 - Martins, Uberlândia - MG, 38400-106, BRA 3,5 km (aprox. 10 minutos)

1. Siga na direção sudeste na Av. João Bernardes de Souza em direção à R. João Justino Fernandes ir 0,4 km
2. Vire à direita na Av. Cesário Crosara ir 0,8 km
3. Faça um retorno na R. Carlos Vilela Marquez ir 0,9 km
4. Vire à direita na Av. João Pessoa ir 1,3 km
5. Vire à direita na R. Cipriano Del Favero O destino estará à direita ir 90 m
6. Chegar em: R. Cipriano Del Favero, 931 - Martins, Uberlândia - MG, 38400-106, BRA

de R. Cipriano Del Favero, 931 - Martins, Uberlândia - MG, 38400-106, BRA a R. Sebastião Vieira da Mota, 43 - Luizote de Freitas, Uberlândia - MG, 38414-352, BRA 7,1 km (aprox. 14 minutos)

1. Siga na direção nordeste na R. Cipriano Del Favero em direção à R. Ricardo Felice ir 41 m
2. Vire à esquerda na R. Ricardo Felice ir 0,1 km
3. Vire à esquerda na Av. Vasconcelos Costa ir 2,7 km
4. Vire à direita na Av. Geraldo Mota Batista ir 0,3 km
5. Vire à esquerda na Av. Brg. Sampaio ir 0,5 km
6. Vire à direita na Av. Asp. Mega ir 1,8 km

7. Na rotatória, pegue a 3ª saída para a Av. José Fonseca E Silva ir 1,5 km
8. Vire à esquerda na R. Genarino Cazabono ir 0,1 km
9. Curva acentuada à direita na R. Sebastião Vieira da Mota O destino estará à esquerda ir 40 m
10. Chegar em: R. Sebastião Vieira da Mota, 43 - Luizote de Freitas, Uberlândia - MG, 38414-352, BRA

de R. Sebastião Vieira da Mota, 43 - Luizote de Freitas, Uberlândia - MG, 38414-352, BRA a Av. Getúlio Vargas, 299 - Martins, Uberlândia - MG, 38400-299, BRA 7,0 km (aprox. 12 minutos)

1. Siga na direção oeste na R. Sebastião Vieira da Mota em direção à Av. José Fonseca E Silva ir 0,1 km
2. Vire à direita na Av. José Fonseca E Silva ir 1,5 km
3. Na rotatória, pegue a 1ª saída para a Av. Asp. Mega ir 2,3 km
4. Vire à esquerda na Av. Imbaúbas ir 9 m
5. Vire à direita na Av. Asp. Mega ir 0,4 km
6. Vire à direita em direção à Av. Getúlio Vargas ir 0,1 km
7. Vire à esquerda na Av. Getúlio Vargas O destino estará à direita ir 2,6 km
8. Chegar em: Av. Getúlio Vargas, 299 - Martins, Uberlândia - MG, 38400-299, BRA

Apesar de ser um pouco mais complexo de fazer a análise do resultado obtido nesse teste do que no teste anterior, pode-se considerar que obteve-se rotas satisfatórias. Através dos mapas pode-se observar que foram considerados as regras de circulação das vias, como vias de mão única e retornos. Nota-se também que o sistema deu preferência por utilizar vias cuja velocidade de circulação pode ser maior, como é caso das avenidas.

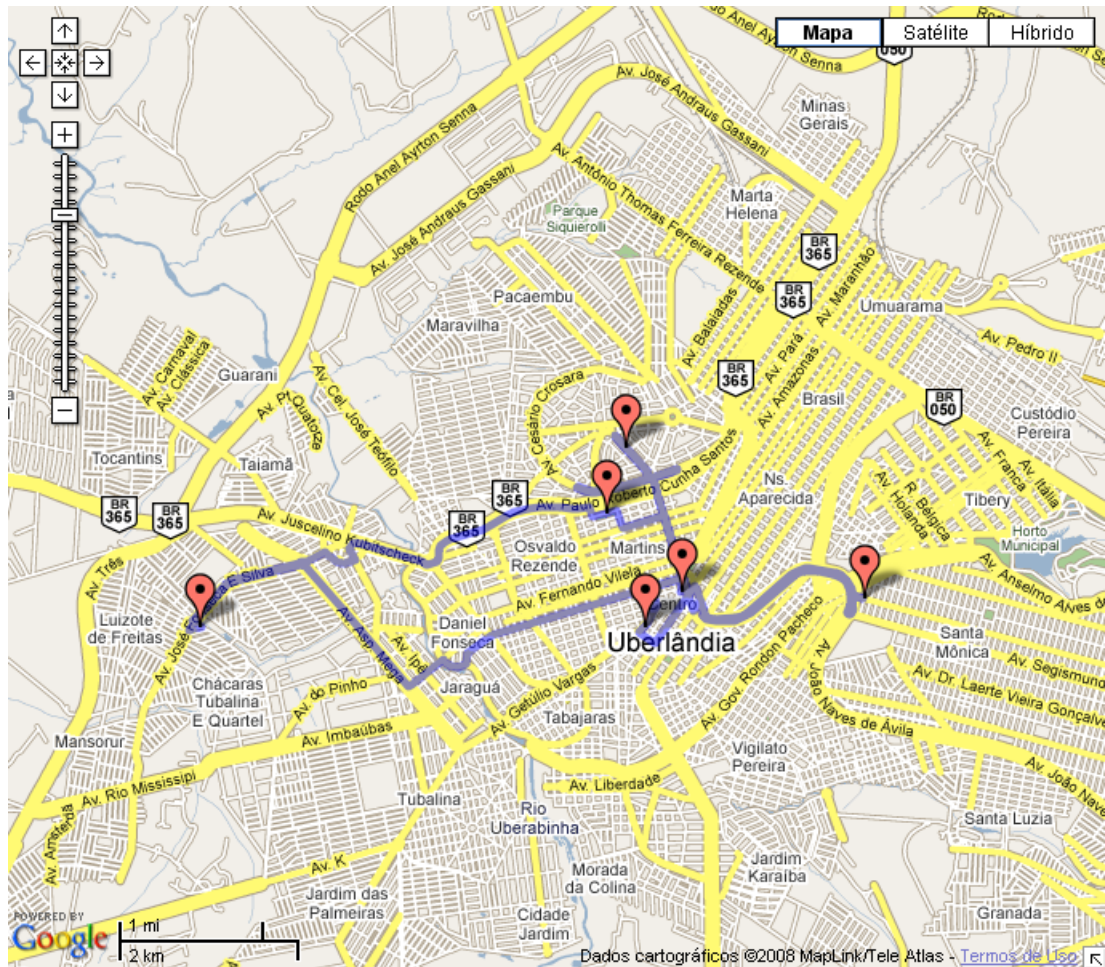


Figura 4.3: Rota gerada para o veículo 1

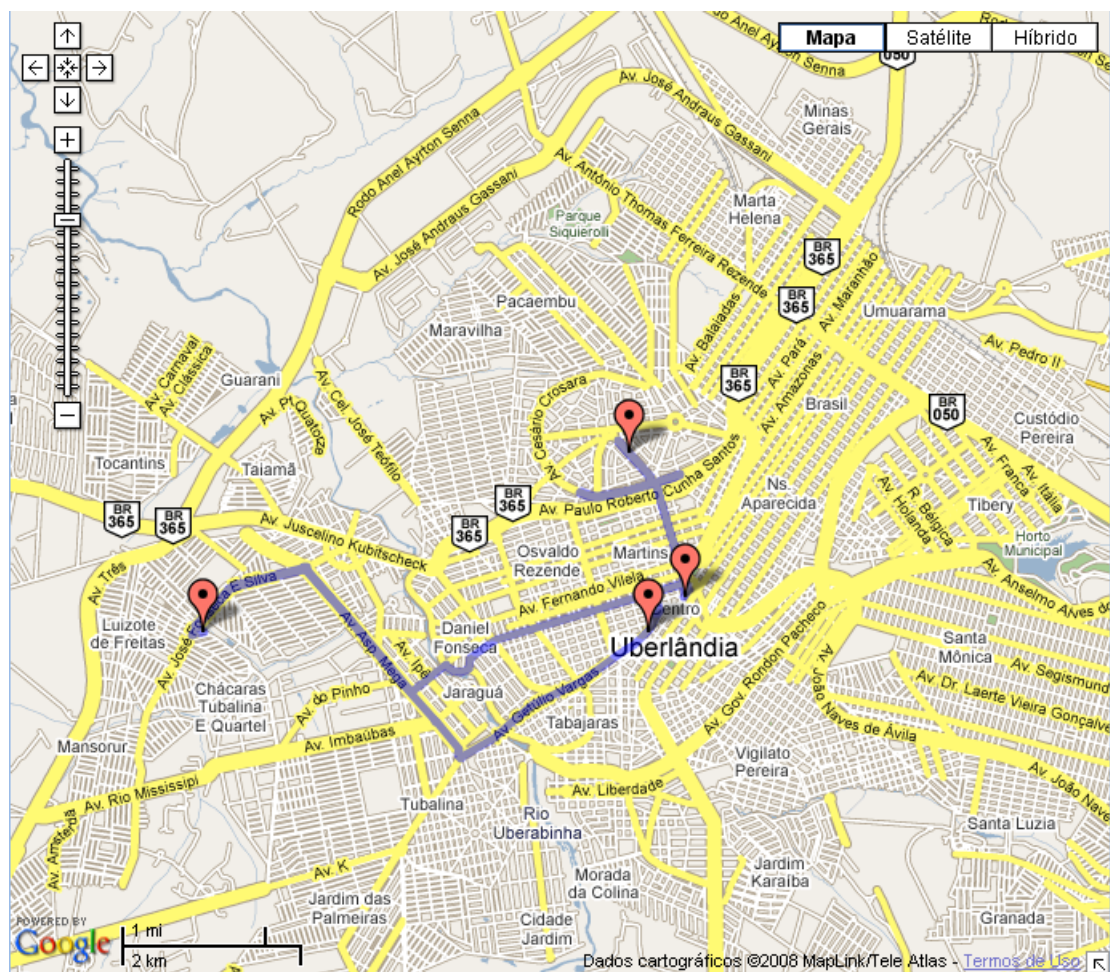


Figura 4.4: Rota gerada para o veículo 2

4.4 Considerações

O tempo de execução do sistema é determinado por dois fatores: desempenho do AG e tempo de resposta dos serviços do GoogleMaps. Na execução do primeiro teste, utilizando-se um processador Intel Core 2 Duo de 1.83 GHz e 2GB de memória DDR2, foram gastos 63 ms. Ao executar novamente o AG com os mesmos parâmetros, o tempo de execução caiu para 42 ms, devido a característica de otimização presente na JVM. O tempo de execução não é um parâmetro adequado para a avaliação do sistema, uma vez que ele depende de requisições feitas a serviços de terceiros, como é o caso do GoogleMaps, cujo tempo de execução é imprevisível pois é afetado por vários fatores como velocidade da rede, tempo de processamento do serviço, *timeout*, etc. Isso faz com que o tempo de execução seja bastante variável. Assim, o gargalo do sistema estará na comunicação com o GoogleMaps e não na execução do AG.

Um fator que pode degradar um pouco o tempo de execução do AG propriamente dito, é o número de pontos para a rota. À medida que o número de pontos aumenta tem-se mais cálculos de distâncias para serem realizados e mais genes nos cromossomos, o que demandará uma maior esforço computacional. Esse mesmo raciocínio aplica-se as rotas múltiplas, pois para cada veículo, tem-se mais um gene no cromossomo.

A hibridização com a técnica 2opt mostrou-se eficiente para resolução do TSP. Com essa técnica o algoritmo é capaz de promover uma melhoria nas soluções já encontradas, de modo a convergir mais rapidamente. Além disso, tem-se uma diminuição no aparecimento de soluções onde os caminhos se cruzam, nos casos em que essa situação pode ser evitada.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Vários trabalhos existentes já comprovaram que a utilização dos Algoritmos Genéticos são eficientes na resolução de problemas de natureza combinatorial. Mais especificamente os problemas baseados no problema Caixeiro Viajante, já demonstrado como um problema de difícil resolução. Porém, a maioria dos trabalhos estudados fazem apenas abordagens teóricas e não propõem soluções aplicáveis em problemas reais.

Apesar da utilização dos AGs na resolução do TSP ser um assunto bastante abundante na literatura, esse trabalho abordou uma nova tendência no campo de estudo das Metaheurísticas, a hibridização de diferentes técnicas. O AG implementado explorou a combinação do AG com a técnica de busca local 2opt. Além de diminuir o tempo de convergência do algoritmo, essa hibridização deixou o algoritmo resultante mais eficiente na determinação de rotas. Além disso, graças a flexibilidade dos AGs, pode-se inserir novas restrições para o problema alterando-se somente a função de aptidão sem comprometer o funcionamento geral do algoritmo.

No caso de trabalhos que abordam o problema da roteirização de veículos, observa-se que na grande maioria dos trabalhos são desenvolvidos protótipos que consideram as localidades como sendo pontos em um plano cartesiano, o custo de deslocamento entre

as localidades, como sendo a distância euclidiana entre elas e as rotas são traçadas em linha reta ligando os pontos. Sabe-se que essa modelagem adotada não aplica-se bem a problemas reais devido a geometria terrestre, às barreiras físicas impostas pelo relevo e pelas vias de circulação.

A *web* vem se consolidando como uma plataforma de desenvolvimento de sistemas e não mais somente como uma agregadora de conteúdo ou um canal de comunicação. Graças a utilização dos recursos da *web 2.0*, esse trabalho conseguiu desenvolver um protótipo com características típicas de softwares comerciais de roteirização. A capacidade de manipular coordenadas geográficas, plotar rotas sobre mapas reais e iterativos e gerar rotas sobre as vias de circulação dos veículos traz o diferencial entre esse trabalho e os demais trabalhos estudados.

As tecnologias de desenvolvimento utilizadas mostraram-se satisfatórias aos propósitos desse trabalho. A linguagem de programação Java possui alto desempenho e facilidade de implementação para aplicações que exijam manipulação de banco de dados e do protocolo HTTP. Além disso, ela conta com ferramentas como a IDE Eclipse que agilizam bastante a etapa de desenvolvimento. O *framework* GWT, também mostrou-se bastante eficiente e ágil para a implementação da parte visual do protótipo, pois foi possível obter funcionalidades típicas de uma aplicação AJAX sem estar sujeito ao baixo rendimento e problemas de incompatibilidade com *browsers* ocasionados pelo desenvolvimento em HTML e JavaScript puros. O protótipo não exigiu muitos recursos do banco de dados, pois lida com poucas tabelas, assim não foi explorado todo o potencial do MySQL, mas é um excelente banco de dados, e, atualmente o mais utilizado para aplicações *web*. Os serviços disponibilizados pela API do Google Maps foram a peça chave para que fosse alcançado um dos principais objetivos desse trabalho, que era processar coordenadas geográficas de localidades e plotar rotas reais sobre o mapa. É importante frisar que todas essas tecnologias utilizadas estão disponíveis para serem utilizadas como software livre.

Assim, esse trabalho foi capaz de unir dois mundos, que muitas vezes mostram-se bem distantes. Um deles é o mundo acadêmico com suas fundamentações teóricas e suas técnicas capazes de serem aplicadas a resolução de inúmeros problemas. O outro é o desenvolvi-

mento de software com suas tecnologias e ferramentas computacionais que atualmente estão presente em todos os segmentos da sociedade para facilitar a vida humana.

5.2 Publicações

- **Utilização de Algoritmos Genéticos Híbridos e Web 2.0 no Desenvolvimento de um Sistema de Roteirização de Veículos**

Rodrigo César de Oliveira

Keiji Yamanaka

Artigo submetido a Revista de Informática Teórica e Aplicada - RITA

5.3 Trabalhos Futuros

Após a implementação de mais funcionalidades e de ser feita uma documentação, as rotinas do AG, serão disponibilizadas Google Code, através do link <http://code.google.com/p/locusgat>, sob forma de uma biblioteca para desenvolvimento de AG para ser utilizada e melhorada por outros desenvolvedores.

Pretende-se melhorar o desempenho e a eficiência do algoritmo de geração com a hibridização de outras técnicas de busca. Deve-se também acrescentar mais restrições ao sistema de roteirização, para que seja capaz de tratar outros problemas encontrados no processo, como tempo de transbordo de carga e capacidade física dos veículos e funcionalidades adicionais como otimização da montagem da carga nos veículos e gerenciamento de armazéns.

O protótipo desenvolvido pode ser complementado com mais funcionalidades e transformado em um sistema completo de roteirização de veículos. Ele pode também ser incorporado a um Sistema Integrado de Gestão Empresarial, que ficaria responsável por todas as etapas das negociações e gerenciamento dos clientes e complementaria o processo de venda com a entrega dos pedidos.

Apresentou-se o protótipo a uma empresa que desenvolve sistemas e presta consultoria

na área de Gestão de Negócios. Ela demonstrou bastante interesse em melhorar o protótipo e integrá-lo aos seus produtos.

Referências Bibliográficas

- [1] MALAQUIAS, N. G. L. *Uso dos Algoritmos Genéticos para a Otimização de Rotas de Distribuição*. Dezembro 2006. Dissertação de Mestrado - Universidade Federal de Uberlândia, Dezembro 2006.
- [2] FREIGHTLINER. Trucks. <http://www.freightlinertrucks.com> – acessado em 01 de setembro de 2008.
- [3] TECHNOLOGIES, R. Routesmart. <http://www.routesmart.com> – acessado em 01 de setembro de 2008.
- [4] MICROANALYTICS. Truckstops. <http://www.bestroutes.com> – acessado em 01 de setembro de 2008.
- [5] EXPRESS, B. Brincity & bronroad. <http://www.brexpress.com.br> – acessado em 01 de setembro de 2008.
- [6] MELO, A. C. S. *Avaliação do uso de sistemas de roteirização de veículos*. 2000. Dissertação de Mestrado - Universidade Federal do Rio de Janeiro, 2000.
- [7] TECH, G. The traveling salesman problem. <http://www.tsp.gatech.edu> – acessado em 01 de setembro de 2008.
- [8] GAREY, M. R.; JOHNSON, D. S. *Computers and intractability: A guide to the theory of np-completeness*. Society for Industrial and Applied Mathematics, 1979.

- [9] GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.
- [10] COLEY, D. A. *Introduction to genetic algorithms for scientists and engineers*. World Scientific, 1998.
- [11] LINDEN, R. *Algoritmos genéticos - uma importante ferramenta da inteligência computacional*. Brasport, 2006.
- [12] TIOBE. Tiobe software. <http://www.tiobe.com> – acessado em 05 de setembro de 2008.
- [13] WIKIPEDIA. Java. [http://pt.wikipedia.org/wiki/Java_\(linguagem_de_programação\)](http://pt.wikipedia.org/wiki/Java_(linguagem_de_programação)) – acessado em 05 de setembro de 2008.
- [14] DEITEL, H. M. *Java how to program*. sexta. ed. Prentice Hall, 2004.
- [15] WIKIPEDIA. Ajax. [http://pt.wikipedia.org/wiki/AJAX_\(programação\)](http://pt.wikipedia.org/wiki/AJAX_(programação)) – acessado em 05 de setembro de 2008.
- [16] WIKIPEDIA. Rich internet application. http://en.wikipedia.org/wiki/Rich_Internet_application – acessado em 05 de setembro de 2008.
- [17] O'REILLY, T. What is web 2.0. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-2.0.html> – acessado em 05 de setembro de 2008.
- [18] GOOGLE. Google web toolkit. <http://code.google.com/webtoolkit> – acessado em 05 de setembro de 2008.
- [19] APACHE. Tomcat. <http://tomcat.apache.org> – acessado em 05 de setembro de 2008.
- [20] GOOGLE. Google maps api. <http://code.google.com/apis/maps> – acessado em 05 de setembro de 2008.
- [21] MICROSYSTEMS, S. Mysql. <http://www.mysql.com> – acessado em 05 de setembro de 2008.

- [22] HIROAKI, S. A fast tsp solver using ga on java. *Proceedings of the Third International Symposium on Artificial Life and Robotics*, v. I, 1998.
- [23] WIKIPEDIA. Problema do caixeiro viajante.
http://pt.wikipedia.org/wiki/Problema_do_caixeiro_viajante – acessado em 14 de junho de 2008.