MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

APLICAÇÕES DE ALGORITMOS GENÉTICOS EM SISTEMAS DE INFORMAÇÕES GEOGRÁFICAS

Adair Santa Catarina

Monografia final do Curso de Introdução ao Geoprocessamento

INPE
São José dos Campos
2004

RESUMO

Neste trabalho realizou-se um estudo sobre a aplicação de algoritmos genéticos em sistemas de informações geográficas. Seis classes de aplicações foram encontradas na pesquisa bibliográfica realizada: otimização em redes, *map labeling*, plano de uso de solos, localização de facilidades, processamento digital de imagens e construção de modelos de interação espacial. Para exemplificar o uso de algoritmos genéticos em sistemas de informações geográficas uma aplicação envolvendo roteamento de veículos foi implementado: o problema de encontrar o caminho mínimo numa rede dado um vértice inicial e um conjunto de pontos intermediários que deverão ser alcançados pelo veículo. O programa desenvolvido proporcionou bons resultados, mas de qualidade inferior àqueles encontrados pelo método determinístico implementado no SPRING v4.0, tanto na qualidade numérica da solução quanto no tempo necessário para obtê-la. Uma característica favorável do algoritmo implementado é que ele é capaz de fornecer rotas alternativas sub-ótimas; assim se o caminho ótimo estiver interditado é possível utilizar outras destas rotas.

SUMÁRIO

Pág.

CAPÍTULO 1 – INTRODUÇÃO

Neste trabalho realizou-se uma pesquisa bibliográfica sobre a utilização de algoritmos genéticos em sistemas de informações geográficas. Para exemplificar este uso desenvolveu-se um programa que utiliza um algoritmo genético num problema relacionado ao geoprocessamento. O problema escolhido pertence a classe dos problemas de roteamento de veículos: encontrar o caminho de custo mínimo.

Os algoritmos genéticos são instrumentos que buscam a solução ótima ou aproximadamente ótima para um problema de otimização. Embora existam soluções determinísticas mais eficientes para o problema escolhido os algoritmos genéticos têm sido utilizados para resolver problemas de roteamento de veículos mais complexos como o problema do caixeiro viajante e roteamento de veículos com janelas de tempo.

Um aspecto favorável ao uso de algoritmos genéticos para encontrar o caminho de custo mínimo é que este mecanismo de busca realiza buscas em paralelo, possibilitando encontrar outras rotas sub-ótimas, dando ao usuário a possibilidade de optar por rotas alternativas, fato que não ocorre com os algoritmos determinísticos clássicos como o algoritmo de Dijkstra.

1.1 Justificativa

Os algoritmos genéticos têm se mostrado eficientes na busca de soluções em grandes espaços de busca, onde os algoritmos determinísticos não conseguem encontrar a solução ótima e tempo viável. Problemas dessa natureza estão, freqüentemente, relacionados à informações geográficas como posicionamento no espaço geográfico.

A construção de um algoritmo genético está intimamente relacionado com o problema a ser resolvido. A escolha da codificação, os operadores genéticos e

os parâmetros genéticos têm de ser estudados para adequá-los ao problema em questão.

Assim, o estudo destes algoritmos aplicados na resolução de problemas envolvendo informações geográficas possibilita conhecer características deste relacionamento fornecendo uma visão mais apurada para o desenvolvimento de um futuro trabalho na área.

1.2 Objetivos

Este trabalho tem por objetivo realizar uma revisão bibliográfica sobre algoritmos genéticos e sua aplicação em sistemas de informações geográficas. Buscou-se, também, verificar a aplicabilidade destes algoritmos num problema em particular, o problema de encontrar um caminho de custo mínimo numa rede viária.

CAPÍTULO 2– OS ALGORITMOS GENÉTICOS

Neste capítulo será apresentada uma breve revisão bibliográfica sobre os algoritmos genéticos. Inicia-se a revisão com o histórico e com a apresentação da estrutura básica destes algoritmos. Posteriormente serão apresentados os principais operadores genéticos e a influência dos parâmetros genéticos sobre o desempenho dos mesmos.

2.1 Introdução aos Algoritmos Genéticos

Os algoritmos genéticos são um tipo de algoritmo de busca que se utiliza do paradigma genético/evolucionário (HOLLAND, 1975). Algoritmos genéticos foram criados com o intuito de imitar alguns dos processos observados na evolução natural das espécies. Os mecanismos que realizam esta evolução ainda não estão completamente compreendidos, mas algumas de suas características já são bem compreendidas e aceitas. A evolução acontece nos cromossomos, que são os elementos orgânicos responsáveis pela codificação genéticas dos seres vivos (DAVIS, 1996). As características e fenômenos específicos desta codificação ainda são objetos de muitas pesquisas. Segundo DAVIS (1996), as principais características gerais da teoria evolucionária que já são amplamente aceitas são:

a) a seleção natural é um processo que atua sobre os cromossomos e, portanto, sobre os seres vivos que eles codificam;

b) a seleção natural é o elo entre cromossomos e a performance das suas estruturas decodificadas. O processo de seleção natural faz com que os cromossomos que codificam estruturas bem sucedidas se reproduzam mais vezes e com maior probabilidade que as estruturas mal sucedidas;

c) o processo de reprodução é o ponto onde a evolução acontece. Mutações podem provocar mudanças nos cromossomos dos filhos, fazendo com que

eles sejam diferentes dos padrões genéticos dos seus pais, e processos de recombinação podem criar diferentes cromossomos para os filhos, pela combinação dos cromossomos dos pais;

d) a evolução biológica não tem memória. Tudo o que se sabe sobre como produzir indivíduos bem adaptados ao seu meio ambiente está contido no seu genoma - o conjunto de cromossomos carregados pelos indivíduos da população atual - e na estrutura dos cromossomos decodificados.

No começo dos anos 70, John Holland, quando pesquisava as características da evolução natural, acreditava que, se estas características fossem adequadamente incorporadas a algoritmos computacionais, poderia produzir uma técnica para solucionar problemas difíceis da mesma forma que a natureza fazia para resolver os seus problemas, ou seja, usando a evolução. Acreditando nisto ele deu início a uma pesquisa sobre algoritmos que manipulavam *strings* de 0 e 1, a qual ele deu o nome de cromossomos. Os algoritmos de Holland realizavam a evolução simulada de populações destes cromossomos. Desta forma, imitando a natureza, seus algoritmos resolviam muito bem o problema de encontrar bons cromossomos, através da manipulação do material contido nos cromossomos.

Outro ponto interessante nas técnicas desenvolvidas por Holland é que, assim como na natureza, estes cromossomos não têm conhecimento nenhum sobre o tipo de problema que estão resolvendo. A única informação que eles dispunham era uma avaliação de cada cromossomo produzido. O objetivo desta avaliação era verificar quais os cromossomos que estavam mais adaptados e, com base nisto, aumentar as suas chances de serem selecionados para a reprodução.

Quando Holland começou os seus estudos sobre estes algoritmos, eles ainda não tinham um nome. Foi apenas quando esta técnica começou a demonstrar o seu potencial que houve a necessidade de se dar um nome adequado e

significativo a ela. Como uma referência às suas origens na biologia, Holland os batizou de algoritmos genéticos. De maneira geral, um algoritmo genético pode ser brevemente descrito através do fluxograma apresentado na figura 2.1.

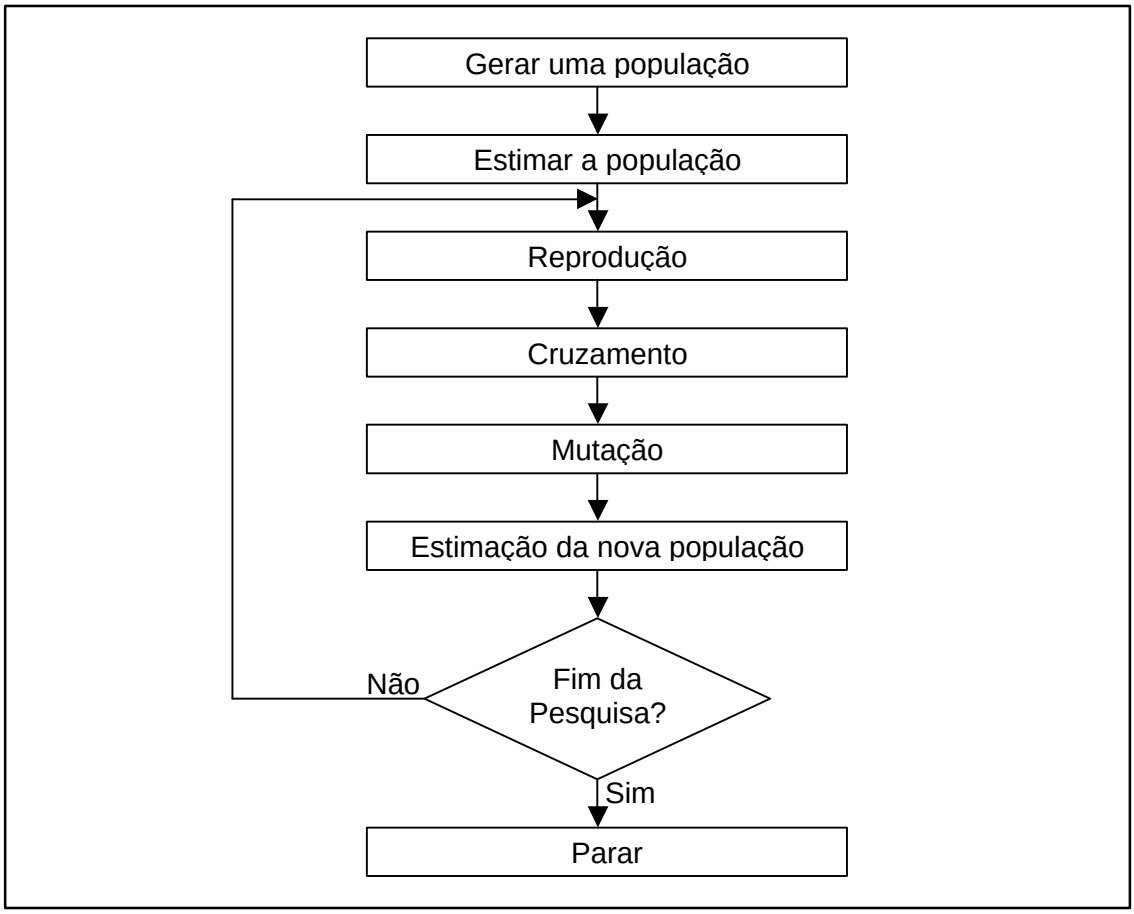

FIGURA 2.1 – Fluxograma que descreve brevemente um algoritmo genético.
Fonte: CORTES (1999)

A técnica usada para codificar as soluções varia de problema para problema e de AG para AG. A codificação clássica usada no trabalho de Holland, e até hoje a mais usada, consistia em usar *strings* de *bits*, mas com o passar do tempo outros pesquisadores apresentaram outras formas de codificação.

A codificação clássica, quando utilizada em problemas que possuem variáveis contínuas e cujas soluções requeridas necessitam boa precisão numérica, torna os cromossomos longos. Para cada ponto decimal acrescentado na precisão, é necessário adicionar 3,3 bits na “string”. (GALVÃO e VALENÇA,

1999)

A conseqüência imediata do aumento da “string”, que representa o cromossomo, é o aumento no tempo necessário para calcular o equivalente decimal deste cromossomo.

Por este motivo, formas não clássicas de codificação dos cromossomos foram desenvolvidas, gerando codificações adequadas para problemas específicos. (HERRERA, LOZANO e VERDEGAY, 1996)

Uma das formas não clássicas de codificação mais utilizada é a codificação real. Esta forma de codificação consiste em representar, num gene ou cromossomo, uma variável numérica contínua através de seu próprio valor real. Um cromossomo pode ser composto por múltiplos genes quando o problema a ser resolvido envolver duas ou mais variáveis.

As primeiras aplicações da codificação real foram propostas por LUCASIUS e KATEMAN (1989) e DAVIS (1989). A partir de então a codificação real tornou-se padrão em problemas de otimização numérica com variáveis contínuas.

CASTRO (1999) afirma que, com certeza, nenhuma forma de codificação funcionaria igualmente bem em todas as situações e que, para cada caso, deve-se fazer uma escolha cuidadosa do tipo de codificação a ser utilizada, pois uma codificação ruim pode não levar ao resultado esperado.

O elemento de ligação entre o AG e o problema a ser resolvido é a função de avaliação. A função de avaliação, chamada de função *fitness*, toma como entrada um cromossomo e retorna um número, ou lista de números, que representam a medida de performance do cromossomo com relação ao problema a ser resolvido. Esta função desempenha no AG o mesmo papel desempenhado pelo meio ambiente na teoria da evolução natural das espécies.

Segundo GOLDBARG e LUNA (2000), os algoritmos genéticos possuem as

seguintes características gerais:

a) Operam em um conjunto de pontos, denominados população, e não a partir de pontos isolados;

b) Trabalham com um conjunto de parâmetros codificados e não com os próprios parâmetros;

c) Necessitam como informação somente o valor de uma função objetivo, denominada função de adaptabilidade ou *fitness*;

d) Usam transições probabilísticas e não regras determinísticas.

2.2 Os Operadores Genéticos

HOLLAND (1975) define três técnicas para criar filhos diferentes dos pais: cruzamento, mutação e inversão. Estes três elementos estão intimamente relacionados no modelo básico de um algoritmo genético; os três fazem a evolução da população acontecer.

A finalidade da seleção em um algoritmo é escolher os elementos da população que devem se reproduzir. Em problemas de maximização, esta escolha deve ser feita de tal forma que dê maior chance de reprodução aos membros da população mais adaptados ao meio ambiente, isto é, àqueles que apresentam um valor da função *fitness* mais elevado. A mais conhecida e utilizada forma de fazer a seleção é a roleta, ou algoritmo Monte Carlo (DAVIS, 1996; MENDES F^{o}, 2004). Na seqüência apresentaremos o funcionamento da seleção através do mencionado algoritmo.

2.2.1 Seleção por Monte Carlo

Na seleção através do algoritmo Monte Carlo, também conhecida como seleção por roleta, cada indivíduo da população é representado numa roleta proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta

aptidão é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa é dada uma porção relativamente menor da roleta. Finalmente, a roleta é girada um determinado número de vezes, dependendo do tamanho da população, e são escolhidos, como indivíduos que participarão da próxima geração, aqueles sorteados na roleta. Um exemplo de aplicação do método da roleta é apresentado na Figura 2.2

S_i	Indivíduo S_i	*Fitness* $f(S_i)$	*Fitness* Relativo
S1	10110	2,23	0,14
S2	11000	7,27	0,47
S3	11110	1,05	0,07
S4	01001	3,35	0,21
S5	00110	1,69	0,11

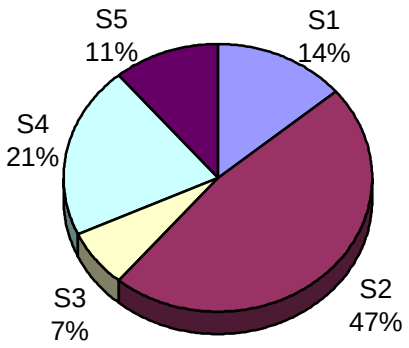

FIGURA 2.2 – Um exemplo de seleção através do algoritmo Monte Carlos ou método da roleta

2.2.2 Elitismo

Para melhorar a convergência dos algoritmos genéticos foi desenvolvida uma técnica chamada elitismo. O elitismo é a técnica mais utilizada para melhorar a convergência destes algoritmos. Ele foi primeiramente introduzido por Kenneth De Jong, em 1975, e é uma adição aos métodos de seleção que força os algoritmos genéticos a reter um certo número de "melhores" indivíduos em cada geração (YEPES, 2004). Tais indivíduos podem ser perdidos se não forem selecionados para reprodução ou se forem destruídos por cruzamento ou mutação. Em outras palavras, o elitismo seleciona os melhores cromossomos de uma população e transporta-os à geração seguinte. Esta técnica consiste basicamente em realizar o processo de seleção em duas etapas:

a) Seleciona-se um elite de *r* membros entre os melhores da população inicial, os quais são incorporados diretamente na população final;

b) O restante da população final é obtida a partir dos (*n* - *r*) elementos restantes da população inicial de tamanho *n*.

Em geral a elite tem um tamanho reduzido, com *r* = 1 ou 2 para um *n* = 50. Quando é utilizada a técnica do elitismo, o algoritmo converge mais rapidamente. Como na natureza, os indivíduos mais aptos podem, além de reproduzir-se mais, ter uma vida mais longa, muitas vezes sobrevivendo de uma geração para a outra e se reproduzindo. O efeito negativo desta estratégia prende-se ao fato de que a população inicial pode convergir para uma população homogênea de super-indivíduos, não explorando outras soluções.

2.2.3 Cruzamento e Mutação

O objetivo final de ambos é fazer com que os cromossomos criados durante o processo de reprodução sejam diferentes dos cromossomos dos pais. O operador de cruzamento é responsável por combinar os cromossomos dos pais na criação dos cromossomos filhos, e o operador de mutação é responsável pela introdução de pequenas mudanças aleatórias nos cromossomos dos filhos. Vários tipos de operadores de cruzamento foram desenvolvidos por vários pesquisadores, alguns adequados a um tipo específico de codificação dos cromossomos, outros com intenção de serem mais genéricos. Mencionaremos aqui apenas os operadores mais comumente utilizados.

O operador mutação de *bit* é aplicável em todas as formas binárias de representação de cromossomos. O processo de mutação de *bit* é bem simples, e normalmente é realizado da seguinte maneira: dada uma certa probabilidade de mutação, normalmente muito baixa e determinada de forma empírica, cada *bit* na *string* do cromossomo é avaliado para saber se este *bit* deverá sofrer uma mutação; caso este *bit* deva sofrer mutação, o seu valor é simplesmente trocado por um valor determinado aleatoriamente entre os valores que podem ser assumidos pelo cromossomo. A Tabela 2.1 mostra 3 cromossomos de

comprimento 4 e os números aleatórios gerados para cada um dos *bits* no cromossomo, os novos *bits* que demonstram as possibilidades de mutação e o resultado final após a mutação. Os número em negrito na coluna **N^{os} Aleatórios** indicam probabilidades muito baixa e, portanto, serão os genes que sofrerão mutação. Os dígitos em negrito na coluna Cromossomo novo são os genes alterados.

TABELA 2.1 – EXEMPLOS DE MUTAÇÃO DE *BIT*

Cromossomo Anterior	N^{os} Aleatórios				Novo *bit*	Cromossomo novo
0011	0,653	**0,001**	0,287	0,373	1	0**1**11
1001	0,721	0,432	0,043	0,840	-	1001
1110	**0,002**	0,076	0,934	0,471	0	**0**110

Quando utiliza-se a codificação em números reais a mutação pode ser realizada de diversas formas: uniforme, gaussiana, *creep*, limite, não-uniforme e não-uniforme múltipla. As três últimas formas de mutação foram propostas por MICHALEWICZ (1994).

A mutação uniforme consiste em substituir o gene selecionado do cromossomo por outro gene gerado aleatoriamente, segundo uma distribuição uniforme, entre os limites mínimo e máximo permitidos. A mutação gaussiana consiste em substituir o gene selecionado por outro gerado a partir de uma distribuição $N(p_i, \sigma^2)$, onde p_i é igual ao valor de gene a ser substituído e a variância é definida pelo pesquisador. GALVÃO e VALENÇA (1999) citam que o valor da variância pode ser diminuído à medida que aumenta o número de gerações do algoritmo genético.

A mutação *creep* consiste em acrescentar ou subtrair um pequeno número aleatório obtido de uma distribuição $N(0, \sigma^2)$ onde a variância assume um valor pequeno. Esta mutação é usada para explorar localmente o espaço de busca.

A mutação não-uniforme consiste na simples substituição de um gene por um

número extraído de uma distribuição não-uniforme. A mutação não-uniforme múltipla consiste em aplicar a mutação não-uniforme em todos os genes do cromossomo selecionado.

O operador de cruzamento em um ponto é a técnica de cruzamento mais simples e a mais utilizada. Esta técnica consiste em dividir os cromossomos selecionados num ponto de sua cadeia, ponto este escolhido aleatoriamente. Após isso, copia-se para os novos cromossomos uma parte de cada um dos cromossomos selecionados - cromossomos pais, formando assim os novos cromossomos - cromossomos filhos. Nas implementações mais tradicionais, é comum um par de cromossomos selecionados dar origem a dois filhos, mas este não é um fator restritivo. A princípio, pode-se criar qualquer quantidade de filhos, desde que, é claro, o número de alelos permita o número desejado de combinações diferentes. A Figura 2.3 apresenta um exemplo do operador de cruzamento em um ponto.

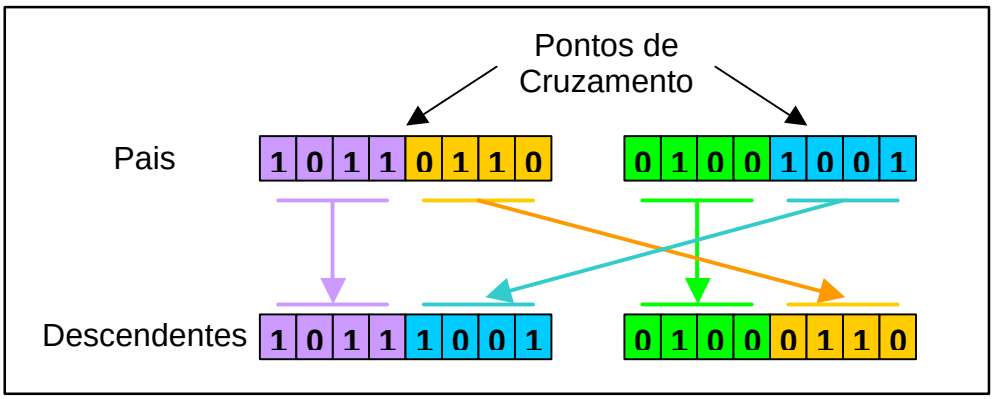

FIGURA 2.3 – Um exemplo do operador de cruzamento em um ponto.
Fonte: YEPES (2004)

Outra técnica de cruzamento, um pouco menos utilizada que a de cruzamento em um ponto, é o cruzamento em múltiplos pontos. Esta técnica divide o cromossomo em vários pontos e recombina-os para formar os filhos, assemelhando-se mais ao processo que ocorre na vida real; possui a vantagem de assegurar uma variedade genética maior.

Nos algoritmos genéticos com codificação real estes operadores de

cruzamento não são adequados, pois apenas trocam os valores dos genes, não criando novos valores. Assim, os operadores de cruzamento aritméticos são mais indicados. Alguns operadores de cruzamento aritméticos são: média (DAVIS, 1996), média geométrica, BLX-α (ESHELMAN e SHAFFER, 1993), aritmético e heurístico (MICHALEWICZ, 1994).

Os cruzamentos média e média geométrica consistem em gerar um novo cromossomo usando a média simples e a média geométrica de dois cromossomos pais, respectivamente.

O cruzamento BLX-α consiste em gerar um novo cromossomo a partir da seguinte expressão:

$$c = p_1 + b\left(p_2 - p_1\right) \tag{2.1}$$

onde c é o novo cromossomo gerado, p_1 e p_2 são os cromossomos pais e $\beta \in U(-\alpha, 1 + \alpha)$. α é um pequeno valor que estende os limites para a definição de c. Caso o cromossomo seja formado por múltiplos genes a eq. 2.1 é aplicada a cada par de genes de p_1 e p_2.

O cruzamento aritmético consiste em gerar dois cromossomos filhos (c_1 e c_2) a partir de dois cromossomos pais (p_1 e p_2), usando a expressão:

$$\begin{aligned} c_1 &= bp_1 + (1-b)p_2 \\ c_2 &= (1-b)p_1 + bp_2 \end{aligned} \tag{2.2}$$

onde $\beta \in U(0, 1)$.

O cruzamento heurístico consiste em gerar um cromossomo filho a partir de uma interpolação linear entre os pais usando a informação da aptidão. Dados dois cromossomos p_1 e p_2 em que p_1 é melhor do que p_2 em termos de aptidão. Então é produzido um cromossomo c da seguinte forma:

$$c = p_1 + r\left(p_1 - p_2\right), \text{ onde } f\left(p_1\right) > f\left(p_2\right) \tag{2.3}$$

onde $r \in U(0, 1)$.

Se compararmos os dois esquemas de reprodução, veremos que no esquema de reprodução sexuada é necessário haver mais de um tipo de indivíduo, estes indivíduos devem ter diferenças significativas em alguns aspectos, e devem desprender uma boa parcela de seu tempo e energia para encontrar um parceiro certo para a reprodução. Isto representa um custo a mais para o indivíduo/algoritmo. Porém, como o esquema de reprodução sexuada parece ter vencido esta guerra, pode-se concluir que este talvez seja um preço pequeno a pagar, comparado aos benefícios que ele traz consigo.

Um benefício proporcionado pela reprodução sexuada é a combinação rápida de características benéficas, o que não é possível no caso da reprodução assexuada. Uma das formas de vida que mais demonstra possuir uma alta capacidade de adaptação reproduz-se assexuadamente, o vírus. O alto poder de adaptação dos vírus vem do fato de que eles são altamente mutáveis, o que pode nos levar a concluir que a capacidade de sofrer mutações também é uma determinante nos organismos naturais. Ainda que não tenhamos cruzamento, se tivermos uma taxa de mutação bastante elevada, nossa população poderá ser capaz de comportar-se como os vírus, mudando sempre para se adaptar ao seu meio ambiente, e reproduzindo-se de forma assexuada.

2.2.4 O Operador de Inversão

O operador de inversão, assim como a mutação, atua sobre um único cromossomo. Ele inverte a ordem dos elementos entre dois pontos escolhidos aleatoriamente no cromossomo. Apesar deste operador ter sido inspirado por processos naturais, ele gera elevada sobrecarga de trabalho, degradando a performance do algoritmo. Na prática, este operador não é muito utilizado (DAVIS, 1996).

2.3 Parâmetros Genéticos

É importante também, analisar de que maneira alguns parâmetros influem no comportamento dos algoritmos genéticos, para que se possa estabelecê-los conforme as necessidades do problema e dos recursos disponíveis.

2.3.1 Tamanho da População

O tamanho da população determina o número de cromossomos na população, afetando o desempenho global e a eficiência dos algoritmos genéticos. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.

2.3.2 Taxa de Cruzamento

Determina a probabilidade em que um cruzamento ocorrerá. Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, a maior parte da população será substituída, e pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

2.3.3 Taxa de Mutação

Determina a probabilidade de ocorrência de uma mutação. Uma baixa taxa de mutação previne a convergência prematura para um ótimo local, possibilitando ao algoritmo explorar melhor todo o espaço de busca. Uma taxa de mutação muito alta faz com que o processo de busca torne-se essencialmente aleatório.

2.3.4 Intervalo de Geração

Controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, podendo ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

CAPÍTULO 3 – APLICAÇÕES DE ALGORITMOS GENÉTICOS

Neste capítulo serão apresentadas aplicações dos algoritmos genéticos que estão relacionadas, de algum modo, com sistemas de informações geográficas.

3.1 Classes de Aplicações de Algoritmos Genéticos em Sistemas de Informações Geográficas

Encontrou-se na literatura 6 classes de aplicações de algoritmos genéticos em sistemas de informações geográficas a seguir citadas.

A primeira classe de aplicações diz respeito aos problemas de otimização em redes, que envolvem, por exemplo, problemas de roteamento de veículos, de distribuição e de projetos de auto-estradas. SKRLEC *et al.* (1996), RONALD e KIRKBY (1998), TANURE e HAMACHER (1998), MAEDA *et al.* (1999), REBELLO e HAMACHER (1999), JHA, McCALL e SCHONFELD (2001), TAN *et al.* (2001), SKOK, SKRLEC e KRAJCAR (2002), TAT e TAO (2003) são exemplos de pesquisadores que realizaram estudos que buscavam solucionar problemas desta classe.

A segunda classe de aplicações é conhecida "*map labeling*". Este problema consiste em espalhar sobre um mapa os nomes de cidades, rios, estradas e demais elementos ali representados, de forma que os textos sejam legíveis, sem sobreposição e estejam alocados próximos às respectivas entidades. Alguns trabalhos desta classe foram desenvolvidos por CHRISTENSEN, MARKS e SHIEBER (1995), ÁLMOS e DMIS (2001), YAMAMOTO e LORENA (2003).

A terceira classe de aplicações são os problemas de concepção de planos de uso de solos. Este problema envolve a distribuição de parcelas de solo à atividades de modo a reduzir riscos ambientais maximizando lucros, por exemplo. Alguns trabalhos envolvendo algoritmos genéticos e sistemas de informações geográficas foram desenvolvidos por BENNETT, ARMSTRONG e

WADE (1996), RONALD e KIRKBY (1998), MATTHEWS *et al.* (1999), MATTHEWS *et al.* (2000) e BJORNSSON, C. e STRANGE, N. (s. d.).

A quarta classe de aplicações caracteriza-se pelos problemas de localização de facilidades. Exemplos deste tipo de problema são a localização de escolas numa cidade de forma que os estudantes não necessitem realizar longos percursos, posicionamento de estações rádio-base celular visando atender a maior área possível com o menor número de antenas, localização de abastecedouros comunitários em regiões agrícolas, etc. Alguns trabalhos realizados desta classe foram desenvolvidos por LORENA e NARCISO (2001), NARCISO e LORENA (2002) e DI CHIARA *et al.* (2003).

A quinta classe de aplicações são relacionadas ao processamento digital de imagens. Algoritmos genéticos são usados em classificação, segmentação e agrupamento em imagens digitais. Os trabalhos de ALLIPI e CUCCHIARA (1992), PERKINS *et al.* (2000), PHAM, WAGNER e CLARK (2001), ALIXANDRINI e CENTENO (2002) são estudos realizados nesta classe de aplicações.

A sexta classe de aplicações é a construção de modelos de interação espacial. Os trabalhos de OPENSHAW e OPENSHAW (1997) e FISCHER e LEUNG (1998) estudam mecanismos baseados em algoritmos genéticos para criar modelos de interação espacial a partir de vários tipos de dados.

3.2 Otimização em Redes

Os algoritmos genéticos aplicados em problemas de otimização em redes visam resolver problemas de roteamento de veículos, de posicionamento de centros de distribuição e projetos de auto-estradas.

SKRLEC *et al.* (1996) propuseram um algoritmo genético para planejar uma rede primária de distribuição de energia. Os autores modificaram um algoritmo genético utilizado para resolução do problema de roteamento de veículos com múltiplas capacidades. Os dados de rede necessários para o processo de

otimização foram extraídos de um sistema de informações geográficas. SKOK, SKRLEC e KRAJCAR (2002) realizaram a integração de um algoritmo genético e um sistema de informações geográficas, baseado no AutoCAD, para solucionar este mesmo tipo de problema.

RONALD e KIRKBY (1998) desenvolveram um algoritmo genético para resolver, simultaneamente, um problema de transporte e de roteamento de veículos. Usaram um esquema de codificação multi-cromossômica, onde cada cromossomo é composto por duas informações. Uma delas referente à rota a ser seguida e outra refere-se ao veículo a ser utilizado em cada trecho da rota. Propõem ainda, os autores, integrar o algoritmo genético desenvolvido com um sistema de informações geográficas que fornecerá os dados espaciais necessários e uma interface de visualização com o usuário.

TANURE e HAMACHER (1998) propõem o uso de um algoritmo genético para solucionar um problema de distribuição dos correios para a zona sul da cidade do Rio de Janeiro. MAEDA *et al.* (1999) e TAN *et al.* (2000) implementaram um algoritmo similar, considerando que os veículos tem uma janela de tempo para atender cada um dos centros de distribuição na rota. Em ambos os trabalhos utiliza-se um sistema de informações geográficas para armazenar a rede sobre a qual o processo de otimização é efetuado.

REBELLO e HAMACHER (1999) implementam o algoritmo genético para solucionar o problema de distribuição dos correios para a zona sul da cidade do Rio de Janeiro. Abordam o problema dividindo-o em dois. Na primeira parte utilizaram um algoritmo genético para criar zonas, agrupando centros de distribuição em *clusters*. Na segunda parte do problema utilizaram outro algoritmo genético para encontrar as melhores rotas entre os *clusters* anteriormente definidos.

JHA e McCALL (2001) e TAT e TAO (2003) utilizaram um sistema de informações geográficas e um algoritmo genético para otimizar o traçado de rodovias. Esta otimização baseou-se em escolher os pontos do trajeto de forma

a minimizar o custo total da obra, considerando informações físicas, naturais e sócio-econômicas das regiões afetadas. Estas informações estão disponíveis no sistema de informações geográficas.

3.3 *"Map Labeling"*

A distribuição da toponímia sobre um mapa, de modo que as informações fiquem legíveis e posicionadas próximas às respectivas entidades, é um problema combinatório. Problemas combinatórios podem ser resolvidos através de um algoritmo genético.

CHRISTENSEN, MARKS e SHIEBER (1995) realizaram um estudo empírico sobre algoritmos para disposição de rótulos em mapas. Compararam os algoritmos de força bruta, guloso, gradiente discreto descendente, de Hirsch, de Zoraster e *simulated annealing*. Este último mostrou-se mais eficiente nos testes realizados. *Simulated annealing* é um algoritmos heurístico estocástico assim como os algoritmos genéticos.

ÁLMOS e DMIS (2001) implementaram um algoritmo genético para dispor rótulos sobre mapas e obtiveram bons resultados com uma estratégia de cromossomo com comprimento variável. Este tipo de cromossomo possibilitou bons resultados mesmo em regiões com muitos rótulos.

YAMAMOTO e LORENA (2003) definiram um algoritmo genético construtivo para o problema de rotular mapas. Em comparações realizadas com 10 diferentes algoritmos este proporcionou melhores resultados mostrando ser adequado para o problema de rotular mapas. Entretanto o tempo computacional para executá-lo é 3,5 vezes superior ao método de busca tabu, que proporciona resultados similares.

3.4 Plano de Uso de Solos

Problemas ambientais são decorrentes de decisões não coordenadas sobre o

uso de solos. Quando os tomadores de decisão não trabalham em conjunto impactos ambientes significativos podem ocorrer.

BENNET, ARMSTRONG e WADE (1996) mostraram como um algoritmo genético pode ser utilizado para construir um elo de ligação entre critérios de decisão e o espaço geográfico, evoluindo-os mutuamente até atingir soluções aceitáveis para problemas ambientais complexos. Agentes inteligentes são utilizados para auxiliar os tomadores de decisão a considerar vários critérios, aprender com os sucessos e com os erros das soluções geradas. Este conhecimento pode ser usado para auxiliar na avaliação de soluções alternativas e gerar soluções melhores para problemas ambientais complexos.

MATTHEWS *et al.* (1999) e MATTHEWS *et al.* (2000) utilizaram algoritmos genéticos para planejar o uso de solos. Consideraram este problema como um problema de alocação espacial. No primeira trabalho otimizou-se o uso dos solos com base num único critério, o econômico. A evolução apresentada no segundo trabalho diz respeito a multi-otimização; neste trabalho além de se considerar o critério econômicos considerou-se também questões como continuidade das áreas.

BJORNSSON e STRANGE (s. d.) aplicam um algoritmo genético para resolver um problema ambiental que ocorre no oeste da Dinamarca: alagar lotes de terras para atividades agrícolas com o menor custo sem com isso causar desvios significativos no teor Fe^{+2} na água, o que alteraria as condições do habitat do salmão. Os autores informam resultados adequados, ou seja, conseguiu-se uma forma mais econômica de alocar as regiões alagáveis sem com isso prejudicar o habitat dos peixes. Informam também que o algoritmo implementado é computacionalmente caro.

3.5 Localização de Facilidades

Os problemas de localização de facilidades caracterizam-se por haver a necessidade de distribuir um número mínimo de facilidades de forma a atingir o

maior número possível de beneficiários

LORENA e NARCISO (2001) criaram um algoritmo genético construtivo para resolver este problema na alocação de recursos no campo e na cidade. O algoritmo desenvolvido é genérico, podendo ser aplicado a qualquer número de facilidades e de usuários e mostrou-se eficaz para resolver estes problemas, tidos como NP-Hard. NARCISO e LORENA (2002) definiram alterações no operador de mutação do algoritmo genético construtivo, tornando-o ainda melhor.

DI CHIARA *et al.* (2003) desenvolveram um *framework* baseado num sistema de informações geográficas e num algoritmo genético para solucionar o problema de otimizar a potência e a localização de estações rádio-base. A avaliação do *framework* desenvolvido, aplicando-o em estudos de caso, mostraram que o mesmo é eficiente e preciso. Relatam ainda que o mesmo pode ser utilizado no planejamento de redes.

3.6 Processamento Digital de Imagens

Algumas das aplicações de algoritmos genéticos em processamento digital de imagens são a classificação, a segmentação e o agrupamento.

ALIPPI e CUCCHIARA (1992) propõem utilizar um algoritmo genético para particionar o espaço, representado numa imagem digital. Encontrar a partição ótima ou sub-ótima dos dados é um processo complexo, pois a problema é naturalmente NP-completo. Os resultados são adequados e os autores ressaltam que a escolha adequada dos operadores de cruzamento e de mutação são fundamentais para que se encontre a solução global ótima.

PERKINS *et al.* (2000) utilizam um algoritmo genético híbrido para realizar a classificação por características em imagens multi-espectrais. O algoritmo é utilizado para encontrar as características espaciais, espectrais ou espaço-espectrais adequadas para o processo de classificação. O número de possíveis

combinações destas características é extremamente grande. Ressaltam os autores que o algoritmo genético combinado com sistemas convencionais de classificação apresenta desempenho superior a estes métodos quando usados isoladamente.

ALIXANDRINI Jr. E CENTENO (2002) utilizam um algoritmo genético para encontrar, por fatiamento, os limites inferiores e superiores em cada banda espectral e nas bandas de textura. Neste caso a tarefa dos algoritmos genéticos consiste em encontrar o conjunto de limiares mais adequados em função das amostras disponíveis. Um segundo algoritmo genético, em conjunto com um classificador de mínima distância, foi utilizado para definir os centros das classes no espaço das variáveis. Relatam os autores que os resultados obtidos indicam que o primeiro algoritmo genético apresenta uma mais fácil convergência e que pode ser utilizada na classificação de imagens multiespectrais com desempenho semelhante a outro classificador, mesmo utilizando um número menor de amostras.

3.7 Construção de Modelos de Interação Espacial

OPENSHAW e OPENSHAW (1997) demonstram como utilizar um algoritmo genético para descobrir relações entre variáveis. Estas variáveis podem representar fenômenos naturais que variam no espaço. Perceber como estas variáveis estão relacionadas é um problema complexo pois matematicamente existem muitas formas de combinar as diferentes variáveis. O algoritmo genético neste caso é utilizado para encontrar quais os coeficientes, operações aritméticas e transcendentes adequados para relacionar as variáveis, ou seja, o modelo matemático que relaciona causas e efeitos. Este algoritmo genético é então chamado de criador de modelos.

FISCHER e LEUNG (1998) apresentam um algoritmo genético com capacidade para maximizar funções complexas. Este algoritmo é utilizado para ajustar a topologia de uma rede neural treinando-a e aumentando sua velocidade de

convergência. Esta rede é, então, utilizada para modelar interação de dados espaciais.

CAPÍTULO 4 – UM ALGORITMO GENÉTICO PARA ROTEAMENTO DE VEÍCULOS

Neste capítulo será apresentado um algoritmo genético que foi implementado para resolver o problema de roteamento de veículos numa rede viária. A cidade escolhida foi Brasília, pois sua rede viária está disponível no banco de dados "Distrito Federal" que acompanha o SPRING v4.0.

Serão apresentados também os resultados obtidos com o respectivo algoritmo comparando-os com os resultados obtidos via algoritmo do custo mínimo, implementado no SPRING v4.0.

Ao final do capítulo serão tecidos alguns comentários sobre as vantagens e desvantagens do algoritmo genético implementado para o problema de roteamento de veículos.

4.1 O Algoritmo Genético Implementado

O algoritmo genético foi implementado usando a IDE Borland Delphi 6, em linguagem Object Pascal. O sistema operacional utilizado foi o Microsoft Windows 2000 Professional e a plataforma de hardware um Pentium IV 2.4GHz com 512 Mb de memória principal.

O software permite realizar a importação de uma rede a partir do formato de intercâmbio de dados geográficos ASCII – SPRING, tipo *network* com coordenadas planas. Esta rede pode então ser exportada para um formato binário e acessada pelo software. O formato deste arquivo é apresentado na figura 4.1.

Os dois primeiros registros do arquivo possuem interpretação especial. Para o primeiro registro, o campo IdLinha equivale ao número de linhas existentes no arquivo e os campos Xi e Yi equivalem às coordenadas do vértice mínimo do retângulo envolvente onde situa-se a rede. Estes mesmos campos do segundo

registro equivalem às coordenadas do vértice máximo deste mesmo retângulo envolvente.

```
IdLinha –  Número inteiro em 32 bits que identifica uma linha
Xi, Yi  -  Números reais em 32 bits correspondentes às coordenadas
           (em m) de um vértice de uma linha
Compr   -  Número real em 32 bits que corresponde ao comprimento da
           linha
Sentido -  Número inteiro em 8 bits que corresponde ao sentido da
           via. 0 = via de mão dupla; 1 = via de mão única sentido
           início-fim; 2 = via de mão única sentido fim-início
```

FIGURA 4.1 – Formato do arquivo binário manipulado pelo software

A estrutura de dados adotada para reter a rede em memória é um grafo. Neste grafo os nós adjacentes estão retidos em listas dinâmicas, o que torna a estrutura mais compacta com relação às matrizes de adjacência.

A rota assinalada no grafo da figura 4.2 inicia-se no vértice 1, chamado de vértice de saída, e passa pelos vértices 2, 6, 7, 8 e 9 que são chamados de intermediários. Os vértices intermediários podem ser percorridos em qualquer ordem. Tanto o vértice inicial quanto os intermediários são definidos pelo operador do sistema. Esta pode ser codificada como 1 – 2 – 5 – 6 – 5 – 7 – 5 – 9 – 10 – 8.

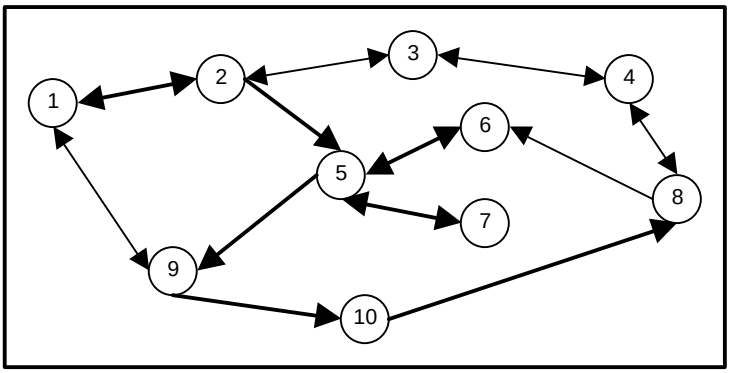

FIGURA 4.2 – Exemplo demonstrando a codificação utilizada pelo algoritmo genético para representar uma rota

A população inicial é gerada aleatoriamente e o número de elementos na população é definido pelo operador. Inicia-se sempre pelo vértice de saída; então escolhe-se aleatoriamente um vizinho a ele conectado. Este processo

repete-se até se ter uma rota constituída com tantos vértices quantos os existentes na rede.

O mecanismo de avaliação calcula então o comprimento de cada uma das rotas aleatoriamente geradas. Um vértice é assinalado como vértice final se ele é o último vértice intermediário a ser atingido na rota. Após o vértice final os demais pontos da rota são ignorados.

Para cada rota é então calculado seu fator de aptidão, através da seguinte fórmula:

$$aptidão = \frac{\left(\frac{1}{Comprimento\ da\ rota}\right)}{3^{NVNAt}} \tag{4.1}$$

onde NVNAt é igual o número de vértices intermediários não atingidos na rota. Assim a função aptidão penaliza aquelas rotas que não passam por todos os vértices intermediários. Posteriormente a população inicial é ordenada pelo fator de aptidão de cada rota.

O processo de evolução pode ou não utilizar o elitismo. Caso use elitismo até 5 elementos podem ser preservados; ou seja, são copiados imediatamente para a população subseqüente.

Os demais elementos são gerados da seguinte forma: usando o método da roleta seleciona-se um elemento da população atual, ou seja, uma rota; para cada vértice desta rota sorteia-se um número aleatório; se este número sorteado for menor que a taxa de mutação então haverá mutação a partir desse vértice; caso contrário apenas copia-se o vértice para a nova rota. A taxa de mutação é um parâmetro também definido pelo operador do sistema.

No caso de haver mutação concatena-se o segmento de rota anterior àquele vértice selecionado para mutação com uma nova rota criada a partir deste vértice. A concatenação resulta então numa nova rota.

O processo de seleção e mutação repete-se até que uma nova população esteja disponível.

Posteriormente esta nova população é avaliada e ordenada pelo fator de aptidão. Verifica-se, então, se um melhor rota foi gerada. Caso isso não ocorra então contamos uma geração estática, ou seja, sem melhoria. Caso contrário zeramos o contador de gerações estáticas. O critério de parada é o contador de gerações estáticas atingir um valor pré-definido pelo operador do sistema.

4.2 Resultados Obtidos

A figura 4.3 apresenta a interface do programa desenvolvido. No painel à esquerda visualiza-se a rede em análise. No painel à direita configuram-se os vértices inicial e intermediários a serem atingidos na rota e os parâmetros genéticos. Ainda no painel à direita podem ser visualizados os resultados obtidos.

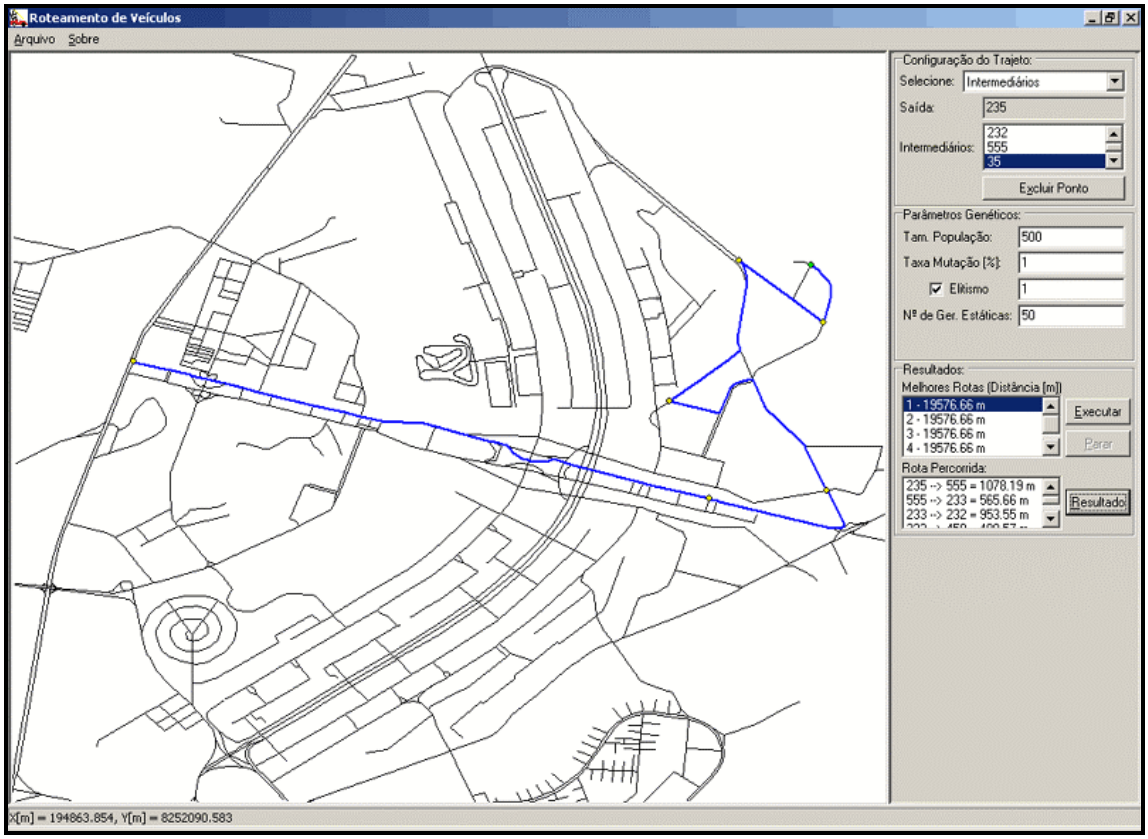

FIGURA 4.3 – Interface do programa desenvolvido

Para testar o algoritmo genético implementado foram realizados 4 casos de teste com a rede apresentada na figura 4.3. O primeiro e o segundo caso consistiram em rotas com dois vértices apenas. O terceiro caso consistiu em selecionar um conjunto de vértices intermediários disperso, caracterizando uma rede ampla. O quarto caso consistiu em selecionar um conjunto de vértices intermediários próximos uns dos outros, caracterizando uma rede restrita.

Para cada um dos casos de teste o programa foi executado 15 vezes. Estas 15 execuções do programa foram subdivididas em grupos de 5 onde variou-se o tamanho da população inicial: 30, 60 e 100 indivíduos.

Os demais parâmetros genéticos foram assim fixados: taxa de mutação = 1%; tamanho da elite = 1 indivíduo; número de gerações estáticas = 200 gerações. Estes valores foram definidos a priori, com base no conhecimento prévio do autor sobre algoritmos genéticos.

4.2.1 Caso 1 – Rede com dois vértices

Os resultados encontrados para o primeiro caso de teste são apresentados na figura 4.4.

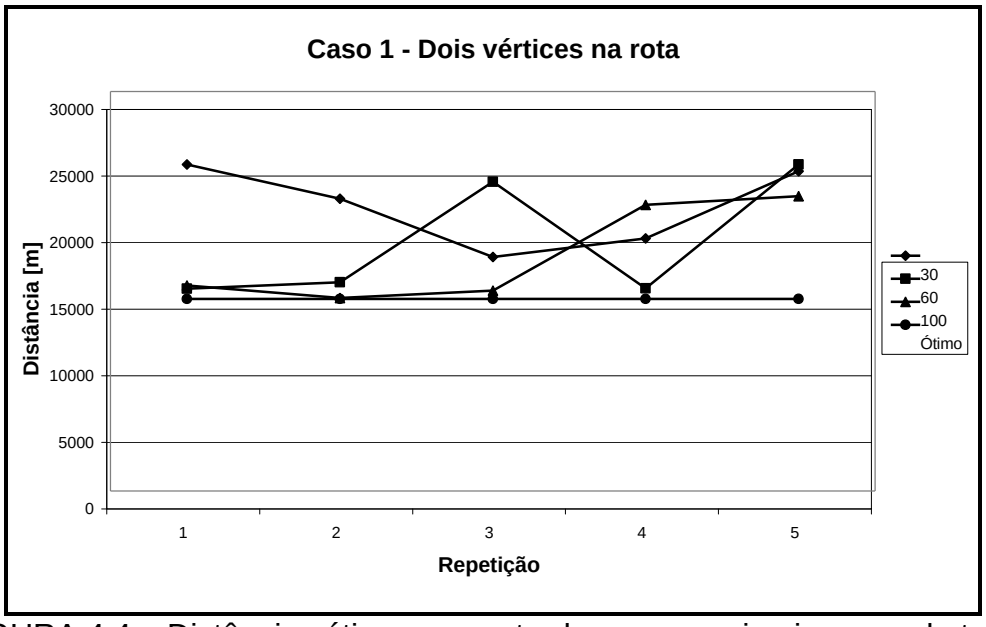

FIGURA 4.4 – Distâncias ótimas encontradas para o primeiro caso de teste

Analisando-se a figura 4.4 percebe-se que uma população inicial com um número maior de indivíduos possibilita encontrar resultados melhores. Entretanto isto não é uma regra rígida pois na quarta execução do programa o melhor resultado foi encontrado com uma população de 60 indivíduos.

A figura 4.5 apresenta o tempo de convergência em cada execução do programa para o caso 1.

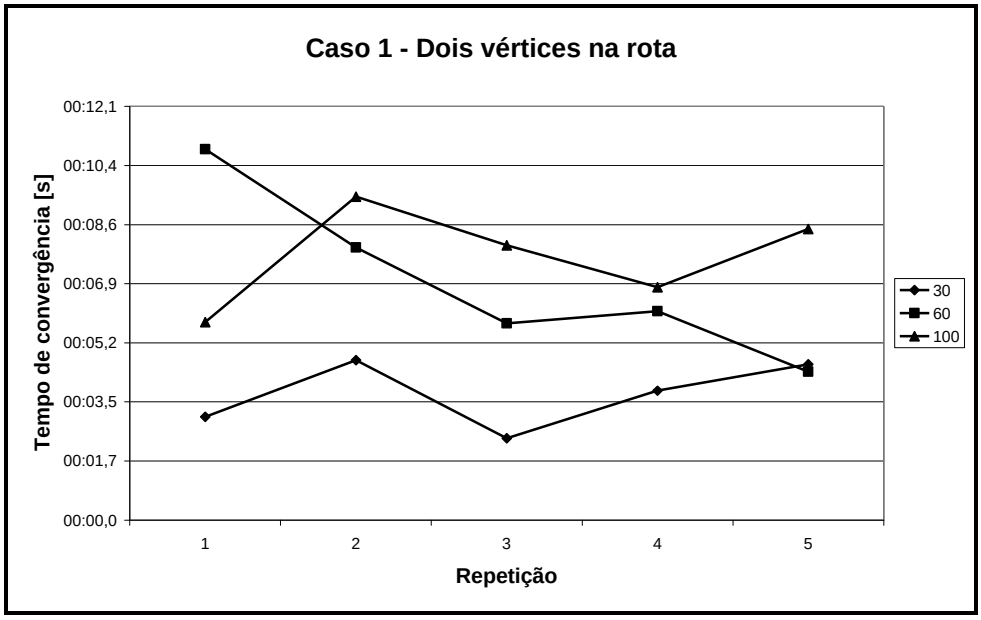

FIGURA 4.5 – Tempo de convergência para o primeiro caso de teste

A figura 4.5 mostra que uma população menor faz com que o programa convirja em tempo menor para uma solução. Já uma população maior tende a fazer com que o programa convirja em tempo maior. Ressalta-se que podem ocorrer exceções neste raciocínio, por exemplo na primeira e na quinta execuções do programa.

4.2.2 Caso 2 – Rede com dois vértices

Os resultados encontrados para o segundo caso de teste são apresentados na figura 4.6 enquanto que o tempo de convergência para cada execução do programa é apresentado na figura 4.7.

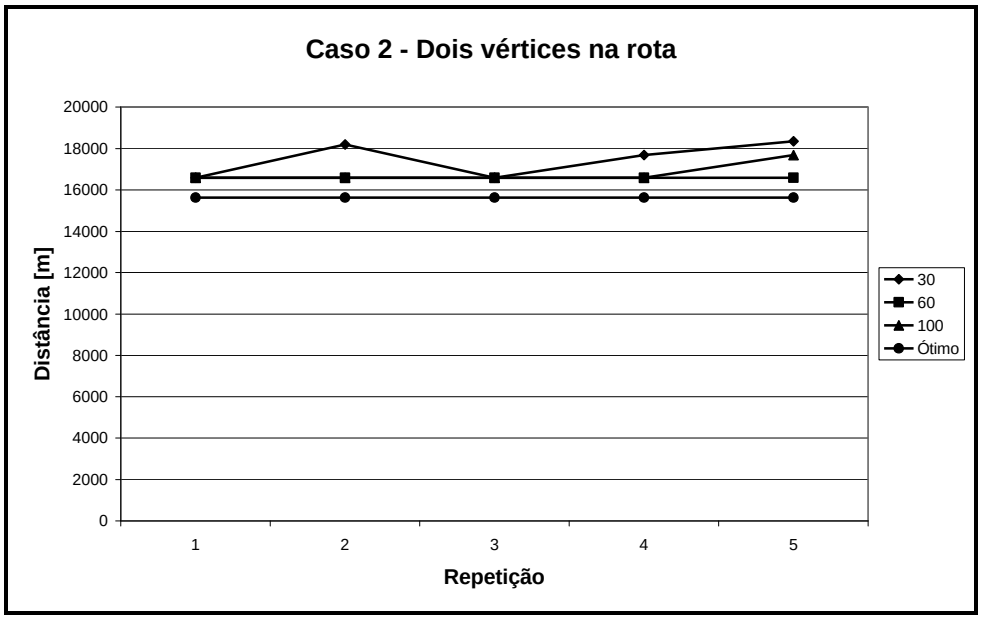

FIGURA 4.6 – Distâncias ótimas encontradas para o segundo caso de teste

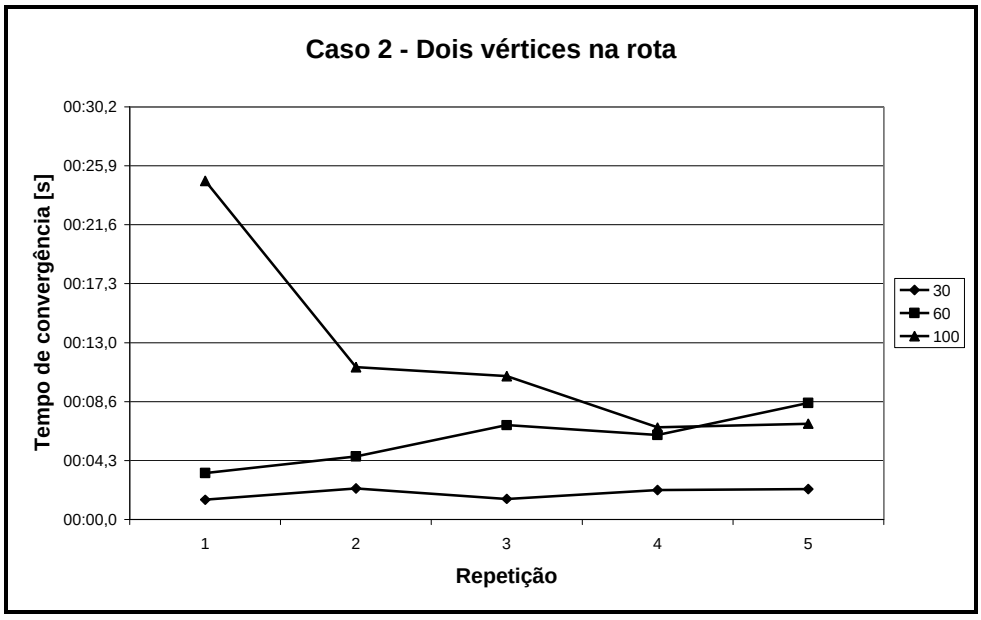

FIGURA 4.7 – Tempo de convergência para o segundo caso de teste

Os resultados obtidos no segundo caso de teste e sintetizados nas figuras 4.6 e 4.7 apresentam comportamento similar aos resultados obtidos no primeiro caso de teste.

4.2.3 Caso 3 – Rede ampla

Os resultados encontrados para o terceiro caso de teste são apresentados na figura 4.8. O tempo de convergência é apresentado na figura 4.9.

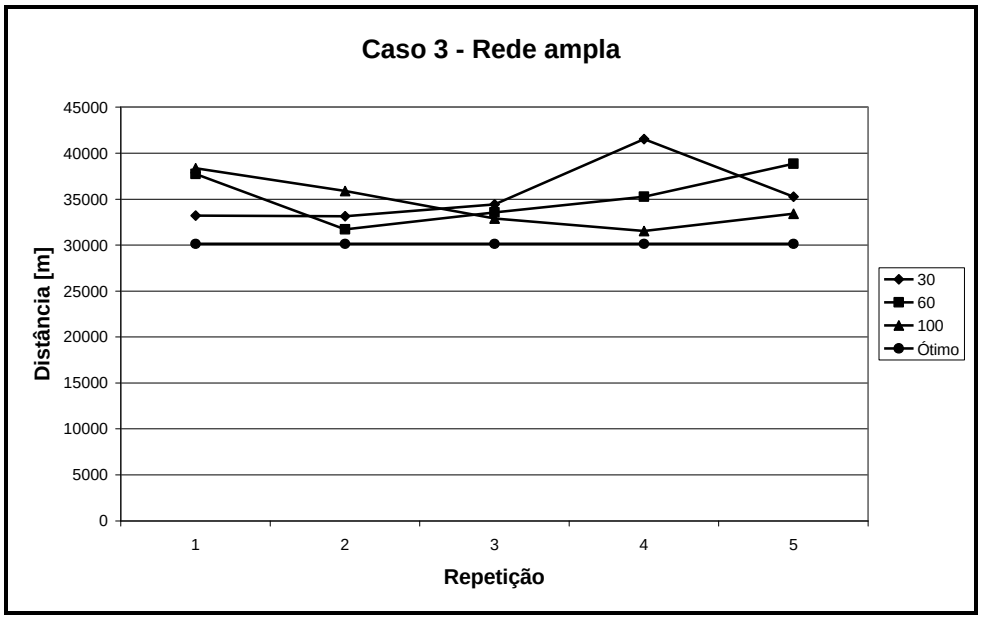

FIGURA 4.8 – Distâncias ótimas encontradas para o terceiro caso de teste

Analisando-se a figura 4.8 percebe-se um comportamento aleatório do programa. Em duas de cinco execuções o melhor resultado foi obtido com populações iniciais que não a maior.

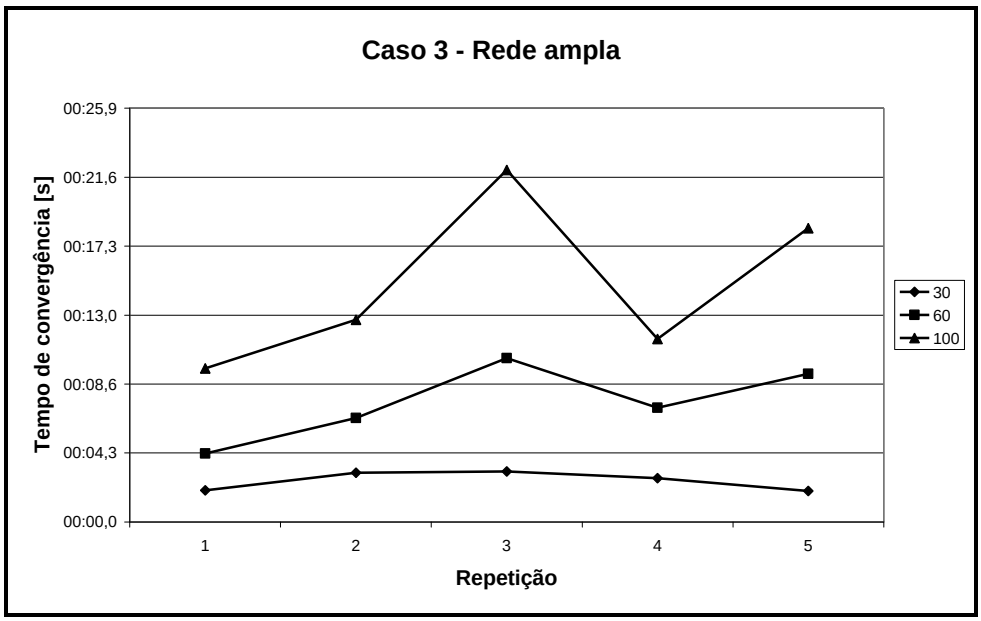

FIGURA 4.9 – Tempo de convergência para o terceiro caso de teste

O comportamento do programa quanto ao tempo de convergência, neste terceiro caso de teste, foi regular. Quanto maior o tamanho da população inicial, maior o tempo de convergência do programa.

4.2.4 Caso 4 – Rede restrita

Os resultados encontrados para o quarto caso de teste são apresentados na figura 4.10. O tempo de convergência é apresentado na figura 4.11.

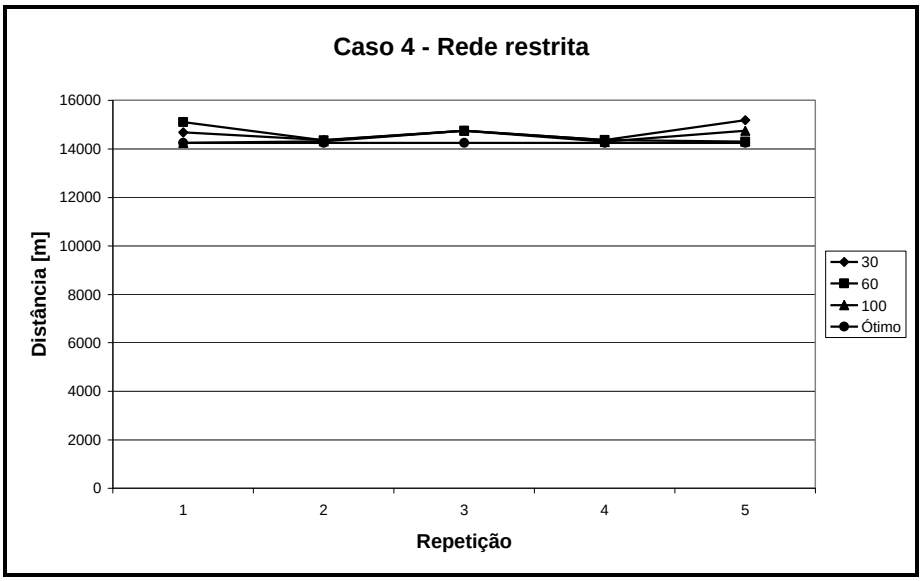

FIGURA 4.10 – Distâncias ótimas encontradas para o quarto caso de teste

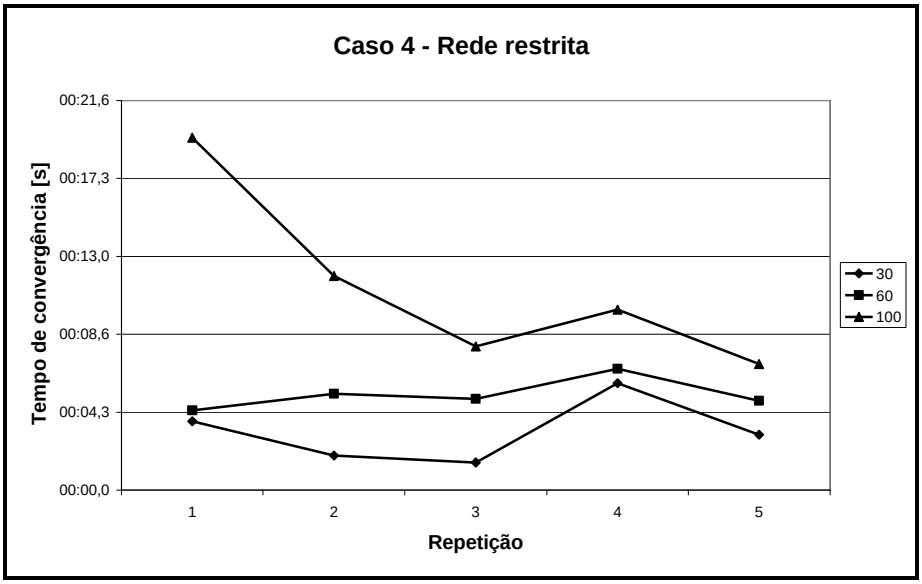

FIGURA 4.11 – Tempo de convergência para o quarto caso de teste

Analisando-se a figura 4.10 percebe-se que uma população inicial maior possibilita encontrar melhores resultados. Particularmente este foi o caso de teste que proporcionou soluções mais próximas da ótima.

Quanto ao tempo de convergência este caso de teste manteve o mesmo comportamento do terceiro caso de teste. Quanto maior a população inicial, maior o tempo de convergência do programa.

CAPÍTULO 5 – CONCLUSÕES

A realização deste trabalho permitiu explorar um aspecto da relação entre algoritmos genéticos e geoprocessamento, no caso o problema de roteamento de veículos com percurso mínimo.

Algumas das conclusões obtidas neste estudo são apresentadas a seguir:

- algoritmos genéticos podem ser utilizados para encontrar soluções ótimas ou aproximadamente ótimas em problemas de roteamento de veículos. Além disso possibilita encontrar outras rotas sub-ótimas alternativas;

- o algoritmo genético, por ser heurístico, não dá garantia de localização da solução ótima; quando for necessária a solução ótima deve-se utilizar um método determinístico;

- o algoritmo genético é melhor recomendado para problemas onde a solução ótima não pode ser obtida por um algoritmo determinístico em tempo polinomial, por exemplo o problema do caixeiro viajante;

- cada problema a ser resolvido exige do operador do sistema conhecimento prévio, pois o bom funcionamento do algoritmo genético depende do ajuste dos parâmetros genéticos e não encontrou-se um padrão que define estes parâmetros;

- uma comparação empírica entre o programa implementado e o SPRING mostrou que o algoritmo genético é mais lento que o método determinístico implementado no SPRING.

Uma série de melhorias podem ser implementadas na tentativa de se obter melhores resultados para o algoritmo genético implementado. Algumas sugestões de trabalhos futuros são apresentadas a seguir:

- modificar a estrutura de codificação das rotas visando possibilitar a

aplicação do operador de cruzamento;

- modificar o algoritmo de modo a estabelecer qual será o vértice final da rota, ou então estabelecer que o vértice inicial será igual ao vértice final;

- vincular a estrutura da rede com a identificação dos objetos que a compõe; isto permitiria a inserção de dados e a visualização do resultados através do nome das ruas e da numeração dos imóveis;

- incorporar as restrições de conversão aos vértices da rota, pois é comum que segmentos de ruas conectados tenham restrições quanto às conversões permitidas;

- estudar mecanismos para tolerar e/ou corrigir pequenos erros de digitalização da malha viária;

- armazenar as rotas ótimas já encontradas, evitando repetir a busca para rotas já conhecidas, ou então utilizar uma rota “similar” como elemento da população inicial numa nova busca;

- implementar um mecanismo que permita direcionar a busca para uma sub-região do espaço de busca, possibilitando que a rota passe por locais de interesse, mesmo que ali não existam pontos intermediários;

- testar o algoritmo implementado com outras redes para corrigir eventuais erros que restaram.

REFERÊNCIAS BIBLIOGRÁFICAS

ALIPPI, C.; CUCCHIARA, R. Cluster partitioning in image analysis classification: a genetic algorithm approach. **Proceedings of CompEuro 92**. p. 139-144. Maio 1992.

ALIXANDRINI Jr., M. J.; CENTENO, J. A. S. Experimento da aplicação de algoritmos genéticos para a integração de dados espectrais e textura na classificação de imagens de alta resolução. **Anais do Simpósio Brasileiro de Geomática**. p. 212-217. Jul. 2002.

ÁLMOS, A.; DMIS, A. V. T. Advanced genetic methods for map labeling. **MBE Miniszimpózium helye**. 2001. <http://www.mit.bme.hu/events/minisy2001/Minisymp2001-Cikkek>. Visitado em 04/05/2004.

BENNETT, D. A.; ARMSTRONG, M. P.; WADE, G. A. Agent mediated consensus-building for environmental problems: a genetic algorithm approach. **Proceedings of Third International Conference on Environmental Modeling and Geographic Information Systems**. 1996. <http://www.uiowa.edu/~geog/faculty/armstrong>. Visitado em 10/05/2004.

BJORNSSON, C.; STRANGE, N. Heuristic allocation of wetlands in GIS. **Esri Library**. s. d. <http://gis.esri.com/library/userconf/proc00/professional/papers/PAP238/p238.htm>. Visitado em 13/05/2004.

CASTRO, J. P. **Um algoritmo evolucionário para geração de planos de rotas**. Florianópolis, 1999. Dissertação (Mestrado em Engenharia da Produção), Universidade Federal de Santa Catarina.

CHRISTENSEN, J.; MARKS J.; SHIEBER, S. An empirical study of algorithms for point-feature label placement. **ACM Transactions on Graphics**. v. 14, n. 3, p. 203-232. Jul. 1995.

CORTES, M. B. S. Introdução à otimização. In: II Jornada de Estatística de Maringá. **Mini-curso: Introdução à otimização**. Maringá : UEM, Departamento de Estatística, 1999.

DAVIS, L. Adapting operator probabilities in Genetic Algorithms. **Proceedings of the Third International Conference on Genetic Algorithms**. San Mateo, 61-69. 1989.

DAVIS, L. **Handbook of Genetic Algorithms**. Reissue edition. Stamford : International Thomson Publishing, 1996.

DI CHIARA, B.; SORRENTINO, R.; STRAPPINI, M.; TARRICONE, L. Genetic optimization of radiobase-station sizing and location using a GIS-based framework: experimental validation. **IEEE Antennas and Propagation Society International Symposium**. v. 2, p.863-866. Jun. 2003.

ESHELMAN, L. J.; SHAFFER, D. J. Real-coded genetic algorithms and interval-schemata. In: WHITLEY, D. L. **Foudations of genetic algorithms 3**. San Mateo, CA: Morgan Kaufman, 1992, p.187-203.

FISCHER, M. M.; LEUNG, Y. A genetic-algorithms based evolutionary computational neural network for modelling spatial interaction data. **Proceedings of 38th Congress of European Regional Science Association**. Set. 1998. <http://www.ersa.org/ersaconfs/ersa98/papers/478.pdf>. Visitado em 14/05/2004.

GALVÃO, C. O.; VALENÇA, M. J. S. **Sistemas inteligentes: aplicações a recursos hídricos e sistemas ambientais**. Porto Alegre: Ed. Universidade/UFRGS/ABRH, 1999.

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização combinatória e programação linear**: modelos e algoritmos. Rio de Janeiro : Campus, 2000.

HERRERA, F.; LOZANO, M.; VERDEGAY, J. L. Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. **Artificial Intelligence Review**. v. 12, n. 4, p. 265-319. 1998.

HOLLAND, J. H. **Adaptation in natural and artificial systems**. Ann Arbor: University of Michigan Press, 1975.

JHA, M. K.; McCALL, C.; SCHONFELD, P. Using GIS, genetic algorithms, and visualization in highway development. **Computer-Aided Civil and Infrastructure Engineering**. v. 16, n. 6, p. 399-414. 2001.

LORENA, L. A. N.; NARCISO, M. G. Uso de algoritmos genéticos em problemas de localização capacitada para alocação de recursos no campo e na cidade. **EMBRAPA - Comunicado Técnico**. v. 10, n.1. Dez. 2001.

LUCASIUS, C. B.; KATEMAN, G. Applications of genetic algorithms in chemometrics. **Proceedings of the Third International Conference on Genetic Algorithms**. p. 170-176. 1989.

MAEDA, O.; NAKAMURA, M.; OMBUKI, B. M.; ONAGA, K. A genetic algorithm approach to vehicle routing problem with time deadlines in geographical information systems. **IEEE International Conference on Systems, Man and Cybernetics**. v. 2, p. 595-600. Out. 1999.

MATTHEWS, K. B.; CRAW, S.; MACKENZIE, S. E.; SIBBALD, A. R. Applying genetic algorithms to land use plannig. **Proceedings of the 18th Annual Conference of the BCS Planning and Scheduling Special Interest Group**. p. 109-115. 1999.

MATTHEWS, K. B.; CRAW, S.; ELDER, S.; SIBBALD, A. R. Evaluating multi-objective land use planning tools using soft systems methods. **Proceedings of the 19th Annual Conference of the BCS Planning and Scheduling Special Interest Group**. p. 109-120. 2000.

MENDES Fº, E. F. **Algoritmos Genéticos**. 2004. <http://www.icmsc.sc.usp.br/~prico/gene1.html>. Visitado em 20/03/2004.

MICHALEWICZ, Z. **Genetic algorithms + data structures = evolution programs**. 3.ed. Springer-Verlag, 1994.

NARCISO, M. G.; LORENA, L. A. N. Uso de algoritmos genéticos em sistema de apoio a decisão para alocação de recursos no campo e na cidade. **Revista Brasileira de Agroinformática**. v. 4, n. 2, p. 172-176. Mar. 2002.

OPENSHAW, S.; OPENSHAW, C. **Artificial intelligence in geography**. West Sussex : John Wiley & Sons Ltd. 1997.

PERKINS, S.; THEILER, J. BRUMBY, S. P.; HARVEY, N. R.; PORTER, R.; SZYMANSKI, J. J.; BLOCH, J. J. GENIE: a hybrid genetic algorithm for feature classification in multi-spectral images. **Proceedings SPIE 4120**. p. 52-62. 2000.

PHAM, T.; WAGNER, M.; CLARK, D. Applications of genetic algorithms, geostatistics and fuzzy c-means clustering to image segmentation. **Proceedings of 2001 Congress on Evolutionary Computation**. v. 2, p. 741-746. Maio 2001.

REBELLO, F. R.; HAMACHER, S. Zoneamento e roteamento de veículos da coleta de correspondência dos correios usando algoritmos genéticos. **Anais do XIII ANPET**. 1999. <http://www.ind.puc-rio.br/artigos/art_hamacher_15.htm>. Visitado em 04/05/2004.

RONALD, S.; KIRKBY, S. Compound optimisation: solving transport and routing

problems with a multi-chromosome genetic algorithm. **IEEE International Conference on Evolutionary Computation**. p. 365-370. Maio 1998.

RONALD, S.; KIRKBY, S. Genetic algorithms for geographical boundary assignment. **IEEE Evolutionary Computation Proceedings**. p. 235-240. Maio 1998.

SKOK, M.; SKRLEC, D.; KRAJCAR, S. Genetic algorithm and GIS enhanced long term planning of large link structured distribution systems. **IEEE Large Engineering Systems Conference on Power Engineering**. p. 55-60. Jun. 2002.

SKRLEC, D.; KRAJCAR, S.; PRIBICEVIC, B.; BLAGAJAC, S. Exploiting the power of genetic algorithm in optimization of distribution networks. **IEEE Electrotechnical Conference**. v. 3, p.1607-1610. Maio 1996.

TAN, K. C.; LEE, L. H.; ZHU, Q. L.; OU, K. Heuristic methods for vehicle routing problem with time windows. **Artificial Inteligence in Engineering**. v. 15, n. 3, p. 281-295. Jul. 2001.

TANURE, C. Z.; HAMACHER, S. Aplicação de algoritmo genético para um problema de distribuição dos correios. **Anais do XII ANPET**. 1998. <http://www.ind.puc-rio.br/artigos>. Visitado em 04/05/2004.

TAT, C. W.; TAO, F. Using GIS and genetic algorithm in highway alignment optimization. **Proceedings of Intelligent Transportation Systems**. v. 2, p.1563-1567. Out. 2003.

YAMAMOTO, M.; LORENA, L. A. N. A constructive genetic approach to point-feature cartographic label placement. **The Fifth Metaheuristics International Conference**. p. 841-847. Ago. 2003.

YEPES, I. **Uma incursão aos algoritmos genéticos**. 2004. <http://www.geocities.com/igoryepes/>. Visitado em 20/05/2004.

APÊNDICE A – CÓDIGO FONTE DO MÓDULO MAIN – AGROUTER

```
unit UMain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Menus, Math, ExtCtrls, ComCtrls;

type
  TLinha = record
     IdLinha    : integer;
     X,
     Y          : real;
     Sentido    : byte;
  end;

  TLDisplay = array of TLinha;

  TRegLinha = record
     IdLinha    : integer;
     Xi,
     Yi,
     Compr      : real;
     Sentido    : byte;
  end;

  TFileLinha = file of TRegLinha;

  TRegAresta = record
     IdAresta, //Id da linha
     IdVi,     //Id do vértice inicial
     IdVf      //Id do vértice final
               : integer;
     Xi,       //coordenada x inicial [m]
     Yi,       //coordenada y inicial [m]
     Xf,       //coordenada x final [m]
     Yf,       //coordenada y final [m]
     Compr     //comprimento da linha [m]
               : real;
     Sentido   //Sentido do segmento. 0 = Ambos os sentidos, 1 = Início-Fim, 2 = Fim-Início
               : byte;
  end;

  TArestas = array of TRegAresta;

  TElemLista = record
     idVertice  : integer;
     Compr      : real;
  end;

  TPIdVertice = ^TNoLista;

  TNoLista = record
     Vertice    : TElemLista;
     Prox       : TPIdVertice;
  end;

  TPDesc = ^TDesc;

  TDesc = record
     Prim, Ult  : TPIdVertice;
     n          : integer;
  end;

  TVertice = record
     X, Y               : real;
     Header             : TPDesc;
  end;

  TGrafo = array of TVertice;

  TBox = record
```

```
   Xi,
   Yi,
   Xf,
   Yf                : real;
end;

TVetInt = array of integer;

TIndFit = record
   Ind               : TVetInt;
   FimRota           : integer;
   Compr,
   FitP              : real;
end;

TPopulacao = array of TIndFit;

TFrmMain = class(TForm)
  ExportDlg: TSaveDialog;
  OpenDlg: TOpenDialog;
  Mnu: TMainMenu;
  MnuArquivo: TMenuItem;
  MnuAbrir: TMenuItem;
  N1: TMenuItem;
  MnuSair: TMenuItem;
  MnuExportar: TMenuItem;
  MnuImportar: TMenuItem;
  ImportDlg: TOpenDialog;
  PanelRede: TPanel;
  PanelParametros: TPanel;
  MemoRede: TMemo;
  PBRede: TPaintBox;
  StBar: TStatusBar;
  GBPontos: TGroupBox;
  Label1: TLabel;
  Label3: TLabel;
  CBSelecao: TComboBox;
  Label4: TLabel;
  EdtSaida: TEdit;
  LBIntermediarios: TListBox;
  BtnExcluir: TButton;
  GroupBox1: TGroupBox;
  Label5: TLabel;
  Label7: TLabel;
  EdtTamPop: TEdit;
  EdtTxMut: TEdit;
  GroupBox3: TGroupBox;
  Label8: TLabel;
  LBDistancias: TListBox;
  BtnResultado: TButton;
  Label9: TLabel;
  EdtNGerEst: TEdit;
  BtnExecutar: TButton;
  MnuSobre: TMenuItem;
  BtnParar: TButton;
  LBRota: TListBox;
  Label2: TLabel;
  EdtTamElite: TEdit;
  CBElitismo: TCheckBox;
  function  EhVizinho(var G: TGrafo; Ponto1, Ponto2: integer): boolean;
  function  EhIntermediario(var Intermed: TVetInt; NIntermed, Ponto: integer): boolean;
  procedure EliminaVaiVolta(var Rota: TIndFit; NIntermed: integer; Intermed: TVetInt);
  procedure EliminaCiclos(var Rota: TIndFit; NIntermed: integer; Intermed: TVetInt);
  function  Pega_Distancia(var G: TGrafo; i, j: integer): real;
  procedure Genetic_Algorithm(var Pop: TPopulacao; Saida, NIntermed, TamPop, TamElite,
            NGerEst: integer; TxMut: real; Elitismo: boolean; Intermed: TVetInt);
  procedure DrawMarcadores(Saida, NIntermed: Integer; var Intermed: TVetInt; B: TBox);
  procedure DrawSeg(var L: TLDisplay; Saida, NIntermed, NSeg, NPRota: integer; var Intermed: TVetInt;
            B: TBox; var Rota: TIndFit);
  procedure DrawRota(var L: TLDisplay; Saida, NIntermed, NLinhas, NPRota: integer;
            var Intermed: TVetInt; B: TBox; var Rota: TIndFit);
  procedure DrawRede(var L: TLDisplay; Saida, NIntermed, NLinhas: integer; var Intermed: TVetInt;
            B: TBox);
  procedure Criar_LDAG(var LDisp: TLDisplay; var Arest: TArestas; var G: TGrafo; var NArest: integer;
            var NVert: integer; var NLin: integer; var B: TBox; Nome: string);
  function  Ver_Edit_Real(Edt: TEdit; Min, Max: real; Mens: string; var Ret: real): boolean;
  function  Ver_Edit_Integer(Edt: TEdit; Min, Max: integer; Mens: string; var Ret: integer): boolean;
```

```
    function  Ver_ListBox_Vazia(LB: TListBox; Mens: string): boolean;
    function  Ver_Edit_Vazia(Edt: TEdit; Mens: string; var Ret: integer): boolean;
    procedure MnuAbrirClick(Sender: TObject);
    procedure MnuSairClick(Sender: TObject);
    procedure MnuImportarClick(Sender: TObject);
    procedure MnuExportarClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PBRedePaint(Sender: TObject);
    procedure PBRedeMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
    procedure PBRedeMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure BtnExcluirClick(Sender: TObject);
    procedure CBSelecaoChange(Sender: TObject);
    procedure MnuSobreClick(Sender: TObject);
    procedure BtnExecutarClick(Sender: TObject);
    procedure BtnPararClick(Sender: TObject);
    procedure LBDistanciasMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
              X, Y: Integer);
    procedure BtnResultadoClick(Sender: TObject);
    procedure CBElitismoClick(Sender: TObject);
    procedure LBRotaClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FrmMain       : TFrmMain;
  G             : TGrafo;
  LDisplay      : TLDisplay;
  Arestas       : TArestas;
  Populacao     : TPopulacao;
  Intermed      : TVetInt;
  NLinhas,                    //Número de pontos de todas as linhas.
  NVertices,                  //Número de vértices do grafo
  NArestas,                   //Número de arestas do grafo
  NIntermed,                  //Número de vértices intermediários
  Saida         : integer;    //Vértice de saída
  Flag          : boolean;    //Sinaliza se o usuário abortou a busca. Usado para parar o AG
  Box           : TBox;       //Coordenadas da área de interesse
  //Retire os comentários a seguir para gerar log
  ArqLog        : Textfile;
implementation
uses Sobre;

{$R *.dfm}
{=============================================================================================}
procedure ClearGrafo(var G: TGrafo);
var
   P,
   Q    : TPIdVertice;
   i    : integer;
begin
   for i := 0 to (NVertices - 1) do begin
      if (G[i].Header <> nil) then begin
         P := G[i].Header^.Prim;

         while (P <> nil) do begin
            Q := P;
            P := P^.Prox;
            dispose(Q);
         end;

         dispose(G[i].Header);
      end;
   end;
   setlength(G, 0);
end;
{=============================================================================================}
procedure InitL(var H: TPDesc);
begin
   new(H);
   H^.Prim := nil;
   H^.Ult  := nil;
   H^.n    := 0;
end;
{=============================================================================================}
```

```
procedure InsL(var H: TPDesc; No : TElemLista);
var
   P, Q    : TPIdVertice;
begin
   new(P);
   P^.Vertice := No;
   P^.Prox := nil;

   if (H^.Prim = nil) then begin
      H^.Prim := P;
   end
   else begin
      Q := H^.Ult;
      Q^.Prox := P;
   end;

   H^.Ult := P;
   inc(H^.n);
end;
{=============================================================================}
function  TFrmMain.EhVizinho(var G: TGrafo; Ponto1, Ponto2: integer): boolean;
var
   Achou        : boolean;
   P            : TPIdVertice;
begin
   Achou := False;
   P := G[Ponto1].Header^.Prim;

   while ((not achou) and (P <> nil)) do begin
      if (P^.Vertice.idVertice = Ponto2) then Achou := True;
      P := P^.Prox;
   end;

   EhVizinho := Achou;
end;
{=============================================================================}
function TFrmMain.EhIntermediario(var Intermed: TVetInt; NIntermed, Ponto: integer): boolean;
var
   Achou     : boolean;
   i         : integer;
begin
   Achou := False;
   i := 0;
   //Verificando se o ponto da rota na posição (Ini + 1) é intermediário
   while ((not Achou) and (i < NIntermed)) do begin
      if (Ponto = Intermed[i]) then Achou := True;
      inc(i);
   end;
   EhIntermediario := Achou;
end;
{=============================================================================}
procedure TFrmMain.EliminaVaiVolta(var Rota: TIndFit; NIntermed: integer; Intermed: TVetInt);
var
   Ini,
   i            : integer;
   Acabou       : boolean;
begin
   //Aqui elimina os pontos do tipo |-- inúteis na rota
   Acabou := False;
   while (Not Acabou) do begin
      Acabou := True;
      Ini := 0;
      while (Ini <= (Rota.FimRota - 2)) do begin
         //Se o ponto atual é igual àquele dois pontos à frente, então são iguais
         if (Rota.Ind[Ini] = Rota.Ind[Ini + 2]) then begin
            //Se não é intermediário remove os pontos inúteis da rota
            if (not EhIntermediario(Intermed, NIntermed, Rota.Ind[Ini + 1])) then begin
               for i := Ini to (Rota.FimRota - 2) do Rota.Ind[i] := Rota.Ind[i + 2];
               dec(Rota.FimRota, 2);
               Acabou := False;
            end
            else inc(Ini);
         end
         else inc(Ini);
      end;
   end;
```

```
   i := 1;
   Rota.Compr := 0;

   repeat
      Rota.Compr := Rota.Compr + Pega_Distancia(G, Rota.Ind[i - 1], Rota.Ind[i]);
      inc(i);
   until (i > Rota.FimRota);
end;
{=============================================================================}
procedure TFrmMain.EliminaCiclos(var Rota: TIndFit; NIntermed: integer; Intermed: TVetInt);
var
   Ini,
   i,
   j,
   k            : integer;
   EhIntermed   : boolean;
begin
   //Aqui elimina as quebras quadradas inúteis
   for i := 15 downto 1 do begin
      Ini := 0;

      while (Ini < (Rota.FimRota - i)) do begin
         j := 1;
         EhIntermed := False;

         while ((not EhIntermed) and (j <= i)) do begin
            EhIntermed := EhIntermediario(Intermed, NIntermed, Rota.Ind[Ini + j]);
            inc(j);
         end;

         if ((not EhIntermed) and (EhVizinho(G, Rota.Ind[Ini], Rota.Ind[Ini + i + 1]))) then begin
            for k := (Ini + 1) to (Rota.FimRota - i) do Rota.Ind[k] := Rota.Ind[k + i];
            dec(Rota.FimRota, i);
         end
         else inc(Ini);
      end;
   end;

   i := 1;
   Rota.Compr := 0;

   repeat
      Rota.Compr := Rota.Compr + Pega_Distancia(G, Rota.Ind[i - 1], Rota.Ind[i]);
      inc(i);
   until (i > Rota.FimRota);
end;
{=============================================================================}
function TFrmMain.Pega_Distancia(var G: TGrafo; i, j: integer): real;
var
   P         : TPIdVertice;
begin
   P := G[i].Header^.Prim;
   while (P^.Vertice.idVertice <> j) do P := P^.Prox;
   Pega_Distancia := P^.Vertice.Compr;
end;
{=============================================================================}
procedure TFrmMain.Genetic_Algorithm(var Pop: TPopulacao; Saida, NIntermed, TamPop, TamElite,
          NGerEst: integer; TxMut: real; Elitismo: boolean; Intermed: TVetInt);
var
   StTime,
   EndTime,
   DecTime      : TDateTime;
   FitMax,
   FitMaxAt,
   SomaFitP     : real;
   NumGerEst,
   NGer         : integer;
   Str_DecTime  : string;
{-----------------------------------------------------------------------------}
   function Pega_Vizinho(var G: TGrafo; j: integer): integer;
   var
      n,
      i         : integer;
      P         : TPIdVertice;
   begin
      n := trunc(random(G[j].Header^.n));
      P := G[j].Header^.Prim;
```

```
      for i := 1 to n do P := P^.Prox;
      Pega_Vizinho := P^.Vertice.idVertice;
   end;
{-------------------------------------------------------------------------------------}
   function Selecao: integer;
   var
      Parcela,
      N_Aleatorio       : real;
      i                 : integer;
   begin
      Parcela := 0;
      i := -1;
      N_Aleatorio := random * SomaFitP;

      repeat
         i := i + 1;
         Parcela := Parcela + Pop[i].FitP;
      until ((Parcela >= N_Aleatorio) or (i = NVertices));

      Selecao := i;
   end;
{-------------------------------------------------------------------------------------}
   procedure Evoluir_Populacao;
   var
      i,
      j,
      k,
      m         : integer;
      NAleat    : real;
      PopC,
      NPop      : TPopulacao;

   begin
      //Dimensionando os vetores da nova populacao
      setlength(PopC, TamPop);
      setlength(NPop, TamPop);

      for i := 0 to (TamPop - 1) do begin
         setlength(PopC[i].Ind, NVertices + 1);
         setlength(NPop[i].Ind, NVertices + 1);
         //Copiando a população para um vetor auxiliar
         for j := 0 to NVertices do PopC[i].Ind[j] := Pop[i].Ind[j];
         PopC[i].Compr := Pop[i].Compr;
         PopC[i].FitP := Pop[i].FitP;
      end;

      //Calculando a soma de todos os fitness
      SomaFitP := 0;
      for i := 0 to (TamPop - 1) do SomaFitP := SomaFitP + Pop[i].FitP;

      i := 0;
      if (Elitismo) then begin
         //Copiando a elite
         for j := 0 to (TamElite - 1) do begin
            for k := 0 to NVertices do NPop[j].Ind[k] := PopC[j].Ind[k];
            NPop[j].Compr := PopC[j].Compr;
            NPop[j].FitP := PopC[j].FitP;
            inc(i);
         end;
      end;

      //i indica a rota posterior à elite. Ou seja, a posição a partir da qual haverá evolução.
      while (i <= (TamPop - 1)) do begin
         m := Selecao; //Selecionou a rota da população atual
         NPop[i].Ind[0] := Saida;

         //Começa de 1 para não mutar o ponto de saída
         for j := 1 to NVertices do begin
            NAleat := random;

            if (NAleat < TxMut) then begin
               //Para todos os pontos a partir do ponto selecionado (j) procura um vizinho para
               //substituí-lo na rota. Substitui também no vetor cópia, pois ainda pode iniciar uma
               //mutação a partir de outra posição j nesta mesma rota
               for k := j to NVertices do begin
                  NPop[i].Ind[k] := Pega_Vizinho(G, NPop[i].Ind[k - 1]);
                  PopC[m].Ind[k] := NPop[i].Ind[k];
```

```
            end;
         end
         else NPop[i].Ind[j] := PopC[m].Ind[j]; //Se não mutar copia o vértice da rota selecionada
      end;

      //Copia novamente para a rota m (rota selecionada) do vetor cópia a rota original
      //Se não copiar, esta rota estaria alterada devido a mutação realizada acima.
      for k := 0 to NVertices do PopC[m].Ind[k] := Pop[i].Ind[k];
      inc(i);
   end;

   //Copiando a nova população para a população atual
   for i := 0 to (TamPop - 1) do begin
      for j := 0 to NVertices do begin
         Pop[i].Ind[j] := NPop[i].Ind[j];
      end;
      //Zerando o comprimento do vetor rota dos vetores auxiliares
      setlength(NPop[i].Ind, 0);
      setlength(PopC[i].Ind, 0);
   end;
   //Zerando o comprimento dos vetores auxiliares
   setlength(NPop, 0);
   setlength(PopC, 0);
end;
{-------------------------------------------------------------------------------------}
   procedure Quicksort (var P: TPopulacao; inic, fim: integer);
      procedure Qsort (inic, fim : integer);
      var
         i,
         j              : integer;
         k              : real;
         Temp           : TIndFit;

      begin
         k := P[(inic + fim) div 2].FitP;
         i := inic;
         j := fim;

         repeat
            while (P[i].FitP > k) do begin
               i := i + 1;
            end;

            while (k > P[j].FitP) do begin
               j := j - 1;
            end;

            if (i <= j) then begin
               Temp := P[i];
               P[i] := P[j];
               P[j] := Temp;
               i := i + 1;
               j := j - 1;
            end;
         until (i > j);

         if (inic < j) then
            Qsort(inic, j);
         if (i < fim) then
            Qsort(i, fim);
      end;
   begin
      Qsort (inic, fim);
   end;
{-------------------------------------------------------------------------------------}
   procedure Avaliar_Populacao;
   var
      i,
      k,
      m,
      NVaAt,
      NVnAt,
      NVAt      : integer;
      Soma      : real;
      VetVer    : array of boolean;
   begin
      NVaAt := NIntermed; //Número de vértices a atingir
```

```
      setlength(VetVer, NVaAt);

      for i := 0 to (TamPop - 1) do begin
         for k := 0 to (NVaAt - 1) do VetVer[k] := False;
         k := 1;
         Soma := 0;

         repeat
            Soma := Soma + Pega_Distancia(G, Pop[i].Ind[k - 1], Pop[i].Ind[k]);

            //Assinala no vetor de verificação que passou pelo vértice intermediário
            for m := 0 to (NIntermed - 1) do begin
               if (Pop[i].Ind[k] = Intermed[m]) then VetVer[m] := True;
            end;

            //Conta quantos vértices intermediários foram atingidos
            NVAt := 0;
            for m := 0 to (NVaAt - 1) do if (VetVer[m]) then inc(NVAt);
            inc(k);
         //para quanto avaliou toda a rota ou quando atingiu todos os vértices intermediários
         until ((k > NVertices) or (NVAt = NVaAt));

         NVnAt := NVaAt - NVAt; //Calcula quantos pontos intermediários na rota não foram atingidos
         Pop[i].Compr := Soma;
         Pop[i].FitP := (1/Soma)/power(3, NVnAt); //Penalização. Divisão por 3^(# pontos não atingidos)
         Pop[i].FimRota := k - 1;
      end;
   end;
{--------------------------------------------------------------------------------------}
   procedure Populacao_Inicial;
   var
      i,
      j         : integer;
   begin
      //Dimensionando o vetor População
      setlength(Pop, TamPop);
      for i := 0 to (TamPop - 1) do begin
         setlength(Pop[i].Ind, NVertices + 1);
         Pop[i].Ind[0] := Saida;
         for j := 1 to NVertices do Pop[i].ind[j] := Pega_Vizinho(G, Pop[i].ind[j - 1]);
      end;
   end;
{--------------------------------------------------------------------------------------}
begin
   //Retire os comentários a seguir para gerar log
   //AssignFile(ArqLog, 'Log.txt');
   //{$i-}
   //Append(ArqLog);
   //{$i+}
   //if (ioresult <> 0) then rewrite(ArqLog);
   //writeln (ArqLog, 'Tam Pop = ', TamPop:0, ' Tx Mut = ', TxMut:0:2, ' Tam Elite = ', TamElite,
   //                 ' NGEst = ', NGerEst);

   randomize;
   TxMut  := TxMut/100;
   NGer := 0;
   NumGerEst := 0;
   StTime := Time;
   //Gerando a população inicial
   Populacao_Inicial;
   Avaliar_Populacao;
   QuickSort(Pop, 0, TamPop - 1);
   FitMaxAt := Pop[0].FitP;

   repeat
      inc(NGer);
      Inc(NumGerEst);
      STBar.SimpleText := Format('Geração = %d   ---   Gerações sem evolução = %d', [NGer, NumGerEst]);
      Evoluir_Populacao;
      Avaliar_Populacao;
      QuickSort(Pop, 0, TamPop - 1);
      FitMax := Pop[0].FitP;

      if (FitMax > FitMaxAt) then begin
         FitMaxAt := FitMax;
         NumGerEst := 0;
      end;
```

```
      //Retire os comentários a seguir para gerar log
      //writeln(ArqLog, NGer:0, ';', FitMax:0:8, ';', Pop[0].Compr:0:2);
      Application.ProcessMessages;
   until ((NumGerEst = NGerEst) or (Flag));

   if (not Flag) then begin
      EndTime := Time;
      DecTime := EndTime - StTime;
      DateTimeToString (Str_DecTime, 'hh:nn:ss:zzz', DecTime);
      STBar.SimpleText := Format('Geração = %d --- Gerações sem evolução = %d ---- Tempo decorrido =
%s',
                           [NGer, NumGerEst, Str_DecTime]);
      //Retire os comentários a seguir para gerar log
      //writeln(ArqLog, Format('Geração = %d --- Gerações sem evolução = %d ---- Tempo decorrido = %s',
      //                    [NGer, NumGerEst, Str_DecTime]));
   end;
end;
{=============================================================================================}
procedure TFrmMain.Criar_LDAG(var LDisp: TLDisplay; var Arest: TArestas; var G: TGrafo;
          var NArest: integer; var NVert: integer; var NLin: integer; var B: TBox; Nome: string);
var
   Arq          : TFileLinha;
   Reg          : TRegLinha;
   E            : TElemLista;
   i,
   j            : integer;
{---------------------------------------------------------------------------------------------}
   function Rotula_Vertices(NUltAresta: integer): integer;
   var
      i, j, k   : integer;

   begin
      k := 0;
      for i := 0 to NUltAresta do begin
         //Rotular os vértices se eles não estiverem rotulados - são vértices ainda não analisados
         if ((Arest[i].IdVi = -1) or (Arest[i].IdVf = -1)) then begin
            if (Arest[i].IdVi = -1) then begin
               Arest[i].IdVi := k;
               inc(k);
            end;
            if (Arest[i].IdVf = -1) then begin
               Arest[i].IdVf := k;
               inc(k);
            end;
         end;

         for j := (i + 1) to NUltAresta do begin
            if (Arest[j].IdVi = -1) then begin
               if ((abs(Arest[i].Xi - Arest[j].Xi) <= 1) and
                   (abs(Arest[i].Yi - Arest[j].Yi) <= 1)) then begin
                  Arest[j].IdVi := Arest[i].IdVi
               end;

               if ((abs(Arest[i].Xf - Arest[j].Xi) <= 1) and
                   (abs(Arest[i].Yf - Arest[j].Yi) <= 1)) then begin
                  Arest[j].IdVi := Arest[i].IdVf;
               end;
            end;

            if (Arest[j].IdVf = -1) then begin
               if ((abs(Arest[i].Xi - Arest[j].Xf) <= 1) and
                   (abs(Arest[i].Yi - Arest[j].Yf) <= 1)) then begin
                  Arest[j].IdVf := Arest[i].IdVi;
               end;

               if ((abs(Arest[i].Xf - Arest[j].Xf) <= 1) and
                   (abs(Arest[i].Yf - Arest[j].Yf) <= 1)) then begin
                  Arest[j].IdVf := Arest[i].IdVf;
               end;
            end;
         end;
      end;
      Rotula_Vertices := k;
   end;
{---------------------------------------------------------------------------------------------}
begin
```

```
   assignfile(Arq, Nome);
   reset(Arq);
   read(Arq, Reg);
   //Os dois primeiros registros contém: Número de arestas e as coordenadas do Box
   NArest := Reg.IdLinha;
   B.Xi := Reg.Xi;
   B.Yi := Reg.Yi;
   read(Arq, Reg);
   B.Xf := Reg.Xi;
   B.Yf := Reg.Yi;
   i := 0;
   j := 0;
   read(Arq, Reg);

   while (not eof(Arq)) do begin
      setlength(LDisp, j + 1);
      //Põe na LDisplay o ponto inicial da linha com um id lido do arquivo de entrada
      LDisp[j].IdLinha := Reg.IdLinha;
      LDisp[j].X := Reg.Xi;
      LDisp[j].Y := Reg.Yi;
      LDisp[j].Sentido := Reg.Sentido;
      inc(j);
      setlength(Arest, i + 1);
      //Esse primeiro ponto da linha é o vértice inicial de uma aresta
      Arest[i].IdVi := -1;
      Arest[i].IdVf := -1;
      Arest[i].IdAresta := Reg.IdLinha;
      Arest[i].Xi := Reg.Xi;
      Arest[i].Yi := Reg.Yi;
      Arest[i].Sentido := Reg.Sentido;
      Arest[i].Compr := Reg.Compr;
      read(Arq, Reg);

      while (Arest[i].IdAresta = Reg.IdLinha) do begin
         setlength(LDisp, j + 1);
         //Adiciona na LDisplay todos os pontos desta mesma linha com o mesmo Id do ponto inicial
         LDisp[j].IdLinha := Reg.IdLinha;
         LDisp[j].X := Reg.Xi;
         LDisp[j].Y := Reg.Yi;
         LDisp[j].Sentido := Reg.Sentido;
         inc(j);
         //Insere o próximo ponto da linha como vértice final da aresta. Repete várias vezes até
inserir,
         //de fato, o último ponto (vértice final da aresta)
         Arest[i].Xf := Reg.Xi;
         Arest[i].Yf := Reg.Yi;
         if (not eof(Arq)) then read(Arq, Reg)
         else inc(Reg.IdLinha);
      end;

      inc(i);
   end;

   NLinhas := j;
   NVert := Rotula_Vertices(NArest - 1); //Colocar um Id único para cada vértice do grafo
   setlength(G, NVert); //Dimensiona o vetor com o número de vértices do grafo

   //Criação do Grafo da Rede
   for i := 0 to (NVert - 1) do begin
      InitL(G[i].Header);
      G[i].X := 0;
   end;

   for i := 0 to (NArest - 1) do begin
      if (G[Arest[i].IdVi].X = 0) then begin //Se Vi da aresta não está na lista, insere-o
         G[Arest[i].IdVi].X := Arest[i].Xi;
         G[Arest[i].IdVi].Y := Arest[i].Yi;
      end;

      if (G[Arest[i].IdVf].X = 0) then begin //Se Vf da aresta não está na lista, insere-o
         G[Arest[i].IdVf].X := Arest[i].Xf;
         G[Arest[i].IdVf].Y := Arest[i].Yf;
      end;

      if (Arest[i].Sentido = 0) then begin //Se sentido é duplo
         E.IdVertice := Arest[i].IdVf;
         E.Compr := Arest[i].Compr;
```

```
         InsL(G[Arest[i].idVi].Header, E); //Insere na lista do Vi o Vf
         E.IdVertice := Arest[i].IdVi;
         E.Compr := Arest[i].Compr;
         InsL(G[Arest[i].IdVf].Header, E); //Insere na lista do Vf o Vi
      end
      else begin
         if (Arest[i].Sentido = 1) then begin //Se sentido é início-fim
            E.IdVertice := Arest[i].IdVf;
            E.Compr := Arest[i].Compr;
            InsL(G[Arest[i].IdVi].Header, E); //Insere na lista do Vi o Vf apenas
         end
         else begin                           //Senão o sentido é fim-início
            E.IdVertice := Arest[i].IdVi;
            E.Compr := Arest[i].Compr;
            InsL(G[Arest[i].IdVf].Header, E); //Insere na lista do Vf o Vi apenas
         end;
      end;
   end;
end;
{=============================================================================}
procedure TFrmMain.DrawMarcadores(Saida, NIntermed: Integer; var Intermed: TVetInt; B: TBox);
var
   i,
   Inter        : integer;
   Xc,
   Yc           : word;
begin
   i := 0;
   PBRede.Canvas.Pen.Color := clBlack;
   PBRede.Canvas.Pen.Width := 1;

   while (i < NIntermed) do begin
      Inter := strtoint(LBIntermediarios.Items.Strings[i]);
      Xc := round((G[Inter].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
      Yc := PBRede.Height - round((G[Inter].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
      PBRede.Canvas.Brush.Color := clYellow;
      PBRede.Canvas.Ellipse(Xc - 3, Yc - 3, Xc + 3, Yc + 3);
      inc(i);
   end;

   if (Saida <> -1) then begin
      Xc := round((G[Saida].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
      Yc := PBRede.Height - round((G[Saida].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
      PBRede.Canvas.Brush.Color := clLime;
      PBRede.Canvas.Ellipse(Xc - 3, Yc - 3, Xc + 3, Yc + 3);
   end;
end;
{=============================================================================}
procedure TFrmMain.DrawSeg(var L: TLDisplay; Saida, NIntermed, NSeg, NPRota: integer;
                           var Intermed: TVetInt; B: TBox; var Rota: TIndFit);
var
   Xc,
   Yc           : word;
   k,
   Np1,
   Np2,
   NAre         : integer;
   Achou        : boolean;
begin
   PBRede.Canvas.Pen.Color := clRed;
   PBRede.Canvas.Pen.Width := 2;
   Np1 := Rota.Ind[NSeg];
   Np2 := Rota.Ind[NSeg + 1];
   k := 0;
   Achou := False;

   while ((k < NArestas) and (not Achou)) do begin
      if (((Arestas[k].IdVi = Np1) and (Arestas[k].IdVf = Np2)) or
          ((Arestas[k].IdVi = Np2) and (Arestas[k].IdVf = Np1))) then Achou := True
      else inc(k);
   end;

   NAre := Arestas[k].IdAresta;
   k := 0;
   while (NAre <> L[k].IdLinha) do inc(k);
   Xc := round((L[k].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
   Yc := PBRede.Height - round((L[k].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
```

```
   PBRede.Canvas.MoveTo(Xc, Yc);
   inc(k);

   while (NAre = L[k].IdLinha) do begin
      Xc := round((L[k].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
      Yc := PBRede.Height - round((L[k].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
      PBRede.Canvas.LineTo(Xc, Yc);
      inc (k);
   end;

   PBRede.Canvas.Pen.Width := 1;
   PBRede.Canvas.Pen.Color := clBlack;
   DrawMarcadores(Saida, NIntermed, Intermed, Box);
end;
{=============================================================================}
procedure TFrmMain.DrawRota(var L: TLDisplay; Saida, NIntermed, NLinhas, NPRota: integer;
          var Intermed: TVetInt; B: TBox; var Rota: TIndFit);
var
   Xc,
   Yc           : word;
   j,
   k,
   Np1,
   Np2,
   NAre         : integer;
   Achou        : boolean;
begin
   PBRedePaint(nil);
   PBRede.Canvas.Pen.Color := clBlue;
   PBRede.Canvas.Pen.Width := 2;
   Np1 := Rota.Ind[0];

   for j := 1 to NPRota do begin
      Np2 := Rota.Ind[j];
      k := 0;
      Achou := False;

      while ((k < NArestas) and (not Achou)) do begin
         if (((Arestas[k].IdVi = Np1) and (Arestas[k].IdVf = Np2)) or
             ((Arestas[k].IdVi = Np2) and (Arestas[k].IdVf = Np1))) then Achou := True
         else inc(k);
      end;

      NAre := Arestas[k].IdAresta;
      k := 0;

      while (NAre <> L[k].IdLinha) do inc(k);

      Xc := round((L[k].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
      Yc := PBRede.Height - round((L[k].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
      PBRede.Canvas.MoveTo(Xc, Yc);
      inc(k);

      while (NAre = L[k].IdLinha) do begin
         Xc := round((L[k].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
         Yc := PBRede.Height - round((L[k].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
         PBRede.Canvas.LineTo(Xc, Yc);
         inc (k);
      end;

      Np1 := Np2;
   end;

   DrawMarcadores(Saida, NIntermed, Intermed, Box);
end;
{=============================================================================}
procedure TFrmMain.DrawRede(var L: TLDisplay; Saida, NIntermed, NLinhas: integer; var Intermed:
TVetInt;
                            B: TBox);
var
   Xc,
   Yc           : word;
   i,
   NL           : integer;
begin
   PBRede.Canvas.Pen.Color := clWhite;
   PBRede.Canvas.Pen.Style := psSolid;
```

```
   PBRede.Canvas.Pen.Width := 1;
   PBRede.Canvas.Brush.Color := clWhite;
   PBRede.Canvas.Brush.Style := bsSolid;
   PBRede.Canvas.Rectangle(0, 0, PBRede.Width, PBRede.Height);
   PBRede.Canvas.Pen.Color := clBlack;
   i := 0;

   while (i < NLinhas) do begin
      NL := L[i].IdLinha;
      Xc := round((L[i].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
      Yc := PBRede.Height - round((L[i].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
      PBRede.Canvas.MoveTo(Xc, Yc);
      inc(i);

      while (NL = L[i].IdLinha) do begin
         Xc := round((L[i].X - B.Xi)/(B.Xf - B.Xi) * PBRede.Width);
         Yc := PBRede.Height - round((L[i].Y - B.Yi)/(B.Yf - B.Yi) * PBRede.Height);
         PBRede.Canvas.LineTo(Xc, Yc);
         inc (i);
      end;
   end;

   DrawMarcadores(Saida, NIntermed, Intermed, Box);
end;
{=============================================================================================}
procedure TFrmMain.MnuAbrirClick(Sender: TObject);
begin
   OpenDlg.FileName := '*.net';

   if (OpenDlg.execute) then begin
      MnuExportar.Enabled := False;
      MemoRede.Clear;
      ClearGrafo(G);
      setLength(LDisplay, 0);
      setLength(Arestas, 0);
      Criar_LDAG(LDisplay, Arestas, G, NArestas, NVertices, NLinhas, Box, OpenDlg.FileName);
      MemoRede.Visible := False;
      PanelRede.Visible := True;
      PanelParametros.Visible := True;
      EdtSaida.Text := '';
      LBIntermediarios.Clear;
      LBDistancias.Clear;
      LBRota.Clear;
      BtnResultado.Enabled := False;
      Saida := -1;
      NIntermed := 0;
      setlength(Intermed, 0);
      PBRedePaint(nil);
   end;
end;
{=============================================================================================}
procedure TFrmMain.MnuSairClick(Sender: TObject);
begin
   FrmMain.Close;
   FrmMain.Release;
   Application.Terminate;
end;
{=============================================================================================}
procedure TFrmMain.MnuImportarClick(Sender: TObject);
begin
   ImportDlg.FileName := '*.spr';

   if (ImportDlg.Execute) then begin
      PanelRede.Visible := False;
      PanelParametros.Visible := False;
      MemoRede.Visible := True;
      MemoRede.Clear;
      MnuExportar.Enabled := True;
      MemoRede.Lines.LoadFromFile(ImportDlg.FileName);
   end;
end;
{=============================================================================================}
procedure TFrmMain.MnuExportarClick(Sender: TObject);
var
   s,
   campo        : string;
   i, j, k      : integer;
```

```
   Arq          : TFileLinha;
   Reg          : TRegLinha;
   Box          : TBox;

begin
   ExportDlg.FileName := '*.net';

   if (ExportDlg.Execute) then begin
      Screen.Cursor := crHourGlass;
      i := 0;
      assignfile (Arq, ExportDlg.FileName);
      rewrite (Arq);
      s := MemoRede.Lines[0];

      //Procurando token NETWORK
      while ((i < MemoRede.Lines.Count) and (s <> 'NETWORK')) do begin
         inc(i);
         s := MemoRede.Lines[i];
      end;

      if (s = 'NETWORK') then begin
         //Procurando token BOX
         while ((i < MemoRede.Lines.Count) and (copy(s, 1, 3) <> 'BOX')) do begin
            inc (i);
            s := MemoRede.Lines[i];
         end;

         if (copy(s, 1, 3) = 'BOX') then begin
            delete(s, 1, 3);
            j := 1;

            //Separando os limites do BOX
            while (s <> '') do begin
               while (not(s[j] in ['0'..'9', '.'])) do delete(s, 1, 1);
               k := pos(',', s);

               if (j < 4) then begin            //Não há vírgula após último limite do box
                  campo := copy(s, 1, k - 1);   //Por isso controla-se o limite com a variável j
                  delete (s, 1, k);
               end
               else begin
                  campo := s;
                  s := '';
               end;

               case j of
                  1 : Box.Xi := strtofloat(campo);
                  2 : Box.Yi := strtofloat(campo);
                  3 : Box.Xf := strtofloat(campo);
                  4 : Box.Yf := strtofloat(campo);
               end;

               inc(j);
            end;

            if (j > 4) then begin
               inc(i);
               s := MemoRede.Lines[i];

               //Procurando token UNITS
               while ((i < MemoRede.Lines.Count) and (copy(s, 1, 5) <> 'UNITS')) do begin
                  inc(i);
                  s := MemoRede.Lines[i];
               end;

               if (copy(s, 1, 5) = 'UNITS') then begin
                  s := copy (s, length(s)-5, 6);

                  //Verificando se a unidade das dimensões é o metro
                  if (s = 'Metros') then begin
                     inc(i);
                     s := MemoRede.Lines[i];

                     //Procurando token INFO_END
                     while ((i < MemoRede.Lines.Count) and (s <> 'INFO_END')) do begin
                        inc(i);
                        s := MemoRede.Lines[i];
```

```
                    end;

                    if (s = 'INFO_END') then begin
                       inc(i);
                       s := MemoRede.Lines[i];
                       j := 0;
                       //Primeiro registro --> IdLinha = No. Arestas; Xi = Box.Xi; Yi = Box.Yi
                       Reg.IdLinha := 0;
                       Reg.Xi := Box.Xi;
                       Reg.Yi := Box.Yi;
                       Reg.Compr := 0;
                       Reg.Sentido := 0;
                       write(Arq, Reg);
                       //Segundo registro --> Xi = Box.Xf; Yi = Box.Yf
                       Reg.Xi := Box.Xf;
                       Reg.Yi := Box.Yf;
                       write(Arq, Reg);

                       while ((i < MemoRede.Lines.Count) and (s <> 'END')) do begin
                          Reg.IdLinha := j;
                          Reg.Sentido := 0;
                          Reg.Compr   := 0;
                          k := pos('   ', s);  //Verificando onde inicia-se o comprimento da linha
                          delete(s, 1, k + 2); //Sabe-se que é depois de 3 espaços em branco
                          k := pos(' ', s);    //Localizando o espaço ao final do comprimento
                          campo := copy(s, 1, k - 1); //Separando o valor do comprimento da linha

                          if (campo[1] = '-') then Reg.Sentido := 2
                          else Reg.Compr := strtofloat(campo);

                          delete(s, 1, k - 1);
                          while (s[1] = ' ') do delete(s, 1, 1);
                          k := pos(' ', s);
                          campo := copy(s, 1, k - 1);

                          if (campo[1] = '-') then Reg.Sentido := 1
                          else Reg.Compr := strtofloat(campo);

                          inc(i);
                          s := MemoRede.Lines[i];

                          while ((i < MemoRede.Lines.Count) and (s <> 'END')) do begin
                             k := pos(' ', s);           //Xi termina no primeiro espaço em branco
                             campo := copy(s, 1, k - 1); //Recortando Xi
                             Reg.Xi := strtofloat(campo);           //Copiando Xi
                             delete(s, 1, k);                       //Buscando Yi
                             while (s[1] = ' ') do delete(s, 1, 1); //Recortando Yi
                             Reg.Yi := strtofloat(s);              //Copiando Yi
                             Reg.IdLinha := j;
                             inc(i);
                             s := MemoRede.Lines[i];
                             write(Arq, Reg);
                          end;

                          inc(i);
                          s := MemoRede.Lines[i];
                          inc(j);
                       end;

                       //Edita primeiro registro para inserir o número de arestas da rede.
                       seek(Arq, 0);
                       Read(Arq, Reg);
                       Reg.IdLinha := j;
                       seek(Arq, 0);
                       write(Arq, Reg);
                       CloseFile(Arq);
                       Screen.Cursor := crDefault;
                       if (s = 'END') then MessageDlg('Arquivo exportado com sucesso.', mtInformation,
[mbOK], 0)
                       else MessageDlg('Símbolo "END" não encontrado.', mtError, [mbOK], 0);
                    end
                    else MessageDlg('Símbolo "END" não encontrado.', mtError, [mbOK], 0);
                 end
                 else MessageDlg('As dimensões das vias não estão em Metros.', mtError, [mbOK], 0);
              end
              else MessageDlg('Símbolo "UNITS" não encontrado.', mtError, [mbOK], 0);
           end
```

```
            else MessageDlg('Limites do BOX incompletos.', mtError, [mbOK], 0);
         end
         else MessageDlg('Símbolo "BOX" não encontrado.', mtError, [mbOK], 0);
      end
      else MessageDlg('Símbolo "NETWORK" não encontrado.', mtError, [mbOK], 0);
   end;
   MnuExportar.Enabled := False;
   Screen.Cursor := crDefault;
end;
{=============================================================================}
procedure TFrmMain.FormCreate(Sender: TObject);
begin
   {Setando as configurações do Windows}
   CurrencyString    := 'R$';
   CurrencyFormat    := 2;
   NegCurrFormat     := 9;
   ThousandSeparator := ',';
   DecimalSeparator  := '.';
   CurrencyDecimals  := 6;
   DateSeparator     := '/';
   ShortDateFormat   := 'dd/mm/yyyy';
   LongDateFormat    := 'dd/mm/yyyy hh:mm:ss';
   TimeSeparator     := ':';
   TimeAMString      := 'AM';
   TimePMString      := 'PM';
   ShortTimeFormat   := 'hh:mm';
   LongTimeFormat    := 'hh:mm';

   {Este comando impede que alterações on-the-fly na configuração
    do Windows sejam percebidas pela aplicação Delphi}
   Application.UpdateFormatSettings := False;

   MemoRede.Align := alClient;
   NLinhas := 0;
   NVertices := 0;
   NArestas := 0;
end;
{=============================================================================}
procedure TFrmMain.PBRedePaint(Sender: TObject);
begin
   DrawRede(LDisplay, Saida, NIntermed, NLinhas, Intermed, Box);
end;
{=============================================================================}
procedure TFrmMain.PBRedeMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
   StBar.SimpleText := Format('X[m] = %.3f', [((X/PBRede.Width)*(Box.Xf-Box.Xi)+Box.Xi)]) + ', ' +
                       Format('Y[m] = %.3f', [((PBRede.Height-Y)/PBRede.Height)*(Box.Yf-
Box.Yi)+Box.Yi]);
end;
{=============================================================================}
procedure TFrmMain.PBRedeMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
          X, Y: Integer);
var
   Xc,
   Yc,
   i,
   j            : integer;
   Achou        : boolean;
begin
   Achou := False;
   i := 0;

   while ((not Achou) and (i < NVertices)) do begin
      Xc := round(((G[i].X - Box.Xi)/(Box.Xf - Box.Xi)) * PBRede.Width);
      Yc := PBRede.Height - round(((G[i].Y - Box.Yi)/(Box.Yf - Box.Yi))*PBRede.Height);
      if ((abs(X - Xc) <= 4) and (abs(Y - Yc) <= 4)) then Achou := True
      else inc(i);
   end;

   if (Achou) then begin
      case (CBSelecao.ItemIndex) of
         0: begin //Ponto de saída
               Saida := i;
               EdtSaida.Text := inttostr(Saida);
            end;
         1: begin //Ponto intermediário
```

```
               Achou := False;
               j := 0;
               if (i <> Saida) then begin
                  while ((not Achou) and (j < LBIntermediarios.Count)) do begin
                     if (i = strtoint(LBIntermediarios.Items[j])) then Achou := True
                     else inc(j);
                  end;
                  if (not achou) then begin
                     LBIntermediarios.Items.Add(inttostr(i));
                     LBIntermediarios.Selected[LBIntermediarios.Count - 1] := True;
                  end;
               end;
            end;
      end;
   end;

   NIntermed := LBIntermediarios.Count;
   setlength(Intermed, NIntermed);
   for i := 0 to (NIntermed - 1) do Intermed[i] := strtoint(LBIntermediarios.Items.Strings[i]);
   PBRedePaint(nil);
end;
{=============================================================================}
procedure TFrmMain.BtnExcluirClick(Sender: TObject);
var
   i    : integer;
begin
   LBIntermediarios.DeleteSelected;
   NIntermed := LBIntermediarios.Count;
   setlength(Intermed, NIntermed);
   for i := 0 to (NIntermed - 1) do Intermed[i] := strtoint(LBIntermediarios.Items.Strings[i]);
   PBRedePaint(nil);
end;
{=============================================================================}
procedure TFrmMain.CBSelecaoChange(Sender: TObject);
begin
   case (CBSelecao.ItemIndex) of
      0: begin
            EdtSaida.Color := clWindow;
            LBIntermediarios.Color := clInactiveCaptionText;
         end;
      1: begin
            EdtSaida.Color := clInactiveCaptionText;
            LBIntermediarios.Color := clWindow;
         end;
   end;
end;
{=============================================================================}
procedure TFrmMain.MnuSobreClick(Sender: TObject);
begin
   FrmSobre.ShowModal;
end;
{=============================================================================}
function TFrmMain.Ver_Edit_Real(Edt: TEdit; Min, Max: real; Mens: string; var Ret: real): boolean;
begin
   Ver_Edit_Real := False;
   try
      Ret := strtofloat(Edt.Text);

      if ((Ret < Min) or (Ret > Max)) then begin
         MessageDlg(Mens, mtError, [mbOK], 0);
         Edt.SetFocus;
      end
      else begin
         Ver_Edit_Real := True;
      end;
   except
      MessageDlg(Mens, mtError, [mbOK], 0);
      Edt.SetFocus;
   end;
end;
{=============================================================================}
function TFrmMain.Ver_Edit_Integer(Edt: TEdit; Min, Max: integer; Mens: string; var Ret: integer):
boolean;
begin
   Ver_Edit_Integer := False;
   try
      Ret := strtoint(Edt.Text);
```

```
      if ((Ret < Min) or (Ret > Max)) then begin
         MessageDlg(Mens, mtError, [mbOK], 0);
         Edt.SetFocus;
      end
      else begin
         Ver_Edit_Integer := True;
      end;
   except
      MessageDlg(Mens, mtError, [mbOK], 0);
      Edt.SetFocus;
   end;
end;
{=============================================================================}
function TFrmMain.Ver_ListBox_Vazia(LB: TListBox; Mens: string): boolean;
begin
   if (LB.Count = 0) then begin
      MessageDlg(Mens, mtError, [mbOK], 0);
      Ver_ListBox_Vazia := False;
      LB.SetFocus;
   end
   else Ver_ListBox_Vazia := True;
end;
{=============================================================================}
function TFrmMain.Ver_Edit_Vazia(Edt: TEdit; Mens: string; var Ret: integer): boolean;
begin
   if (Edt.Text = '') then begin
      MessageDlg(Mens, mtError, [mbOK], 0);
      Ver_Edit_Vazia := False;
      Edt.SetFocus;
   end
   else begin
      Ret := strtoint(Edt.Text);
      Ver_Edit_Vazia := True;
   end;
end;
{=============================================================================}
procedure TFrmMain.BtnExecutarClick(Sender: TObject);
var
   TamPop,
   TamElite,
   NGerEst,
   i            : integer;
   TxMut        : real;
{-----------------------------------------------------------------------------}
   procedure Quicksort (var P: TPopulacao; inic, fim: integer);
      procedure Qsort (inic, fim : integer);
      var
         i,
         j              : integer;
         k              : real;
         Temp           : TIndFit;

      begin
         k := P[(inic + fim) div 2].Compr;
         i := inic;
         j := fim;

         repeat
            while (P[i].Compr < k) do begin
               i := i + 1;
            end;

            while (k < P[j].Compr) do begin
               j := j - 1;
            end;

            if (i <= j) then begin
               Temp := P[i];
               P[i] := P[j];
               P[j] := Temp;
               i := i + 1;
               j := j - 1;
            end;
         until (i > j);

         if (inic < j) then
```

```
            Qsort(inic, j);
          if (i < fim) then
            Qsort(i, fim);
       end;
    begin
       Qsort (inic, fim);
    end;
{-------------------------------------------------------------------------------}
begin
    TamElite := 0;
    if (Ver_Edit_Vazia(EdtSaida, 'Selecione um ponto de Saída.', Saida) and
        Ver_ListBox_Vazia(LBIntermediarios, 'Escolha pelo menos um ponto intermediário.') and
        Ver_Edit_Integer(EdtTamPop, 5, 1000, 'Digite um valor inteiro entre 5 e 1000.', TamPop) and
        Ver_Edit_Real(EdtTxMut, 0, 5, 'Digite um valor real entre 0 e 5.', TxMut) and
        Ver_Edit_Integer(EdtTamElite, 1, 5, 'Digite um valor inteiro entre 1 e 5.', TamElite) and
        Ver_Edit_Integer(EdtNGerEst, 1, 2000, 'Digite um valor inteiro entre 1 e 2000.', NGerEst)) then
begin

        setlength(Populacao, 0);
        BtnParar.Enabled := True;
        LBDistancias.Clear;
        LBRota.Clear;
        BtnResultado.Enabled := False;
        Flag := False;
        Genetic_Algorithm(Populacao, Saida, NIntermed, TamPop, TamElite, NGerEst, TxMut,
                          CBElitismo.Checked, Intermed);

        if (not Flag) then begin
           //Reordena as 5 melhores rotas pela distância percorrida
           Quicksort (Populacao, 0, 4);
           //Elimina vértices improdutivos 5 melhores rotas
           for i := 0 to 4 do begin
              EliminaCiclos(Populacao[i], NIntermed, Intermed);
              EliminaVaiVolta(Populacao[i], NIntermed, Intermed);
              EliminaCiclos(Populacao[i], NIntermed, Intermed);
              EliminaVaiVolta(Populacao[i], NIntermed, Intermed);
           end;

           //Retire os comentários a seguir para gerar log
           //writeln(ArqLog, 'FIM ;', Populacao[0].FitP:0:8, ';', Populacao[0].Compr:0:2);
           //CloseFile(ArqLog);
           //Exibe as 5 melhores rotas encontradas
           for i := 0 to 4 do
              LBDistancias.Items.Strings[i] := format('%d - %.2f m', [i + 1, Populacao[i].Compr]);
        end;

        BtnParar.Enabled := False;
    end;
end;
{===============================================================================}
procedure TFrmMain.BtnPararClick(Sender: TObject);
begin
    Flag := True;
end;
{===============================================================================}
procedure TFrmMain.LBDistanciasMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
          X, Y: Integer);
begin
    if (LBDistancias.ItemIndex <> -1) then BtnResultado.Enabled := True
    else BtnResultado.Enabled := False;
end;
{===============================================================================}
procedure TFrmMain.BtnResultadoClick(Sender: TObject);
var
    NPRota,
    j            : integer;
    Dist         : real;
begin
    NPRota := LBDistancias.ItemIndex;
    LBRota.Clear;

    if (NPRota <> - 1) then begin
       DrawRota(LDisplay, Saida, NIntermed, NLinhas, Populacao[NPRota].FimRota, Intermed,
                Box, Populacao[NPRota]);

       for j := 1 to Populacao[NPRota].FimRota do begin
          Dist := Pega_Distancia(G, Populacao[NPRota].Ind[j - 1], Populacao[NPRota].Ind[j]);
```

```
         LBRota.Items.Add(Format('%d --> %d = %.2f m', [Populacao[NPRota].Ind[j - 1],
                                  Populacao[NPRota].Ind[j], Dist]));
      end;
   end;
end;
{=============================================================================}
procedure TFrmMain.CBElitismoClick(Sender: TObject);
begin
   if (CBElitismo.Checked) then begin
      EdtTamElite.Color := clWindow;
      EdtTamElite.Enabled := True;
   end
   else begin
      EdtTamElite.Enabled := False;
      EdtTamElite.Color := clInactiveCaptionText;
   end;
end;
{=============================================================================}
procedure TFrmMain.LBRotaClick(Sender: TObject);
var
   NPRota,
   NSeg       : integer;
begin
   NPRota := LBDistancias.ItemIndex;
   NSeg := LBRota.ItemIndex;

   if (NPRota <> -1) then begin
      DrawRota(LDisplay, Saida, NIntermed, NLinhas, Populacao[NPRota].FimRota, Intermed, Box,
               Populacao[NPRota]);
      DrawSeg(LDisplay, Saida, NIntermed, NSeg, Populacao[NPRota].FimRota, Intermed, Box,
              Populacao[NPRota]);
   end;
end;
{=============================================================================}
end.
```