Métodos Heurísticos

Teoria e Implementações

Tutorial desenvolvido por:
Fabrício Bueno

IFSC/Araranguá, 2009

Sumário

Métodos Heurísticos....1
1Algoritmos Genéticos....2
1.1Terminologia....3
1.2Passos do algoritmo....4
1.2.1Seleção....5
1.2.2Cruzamento....6
1.2.3Mutação....7
1.3Variações das Etapas dos Algoritmos Genéticos....8
1.3.1Outras Estratégias de Crossover....8
1.3.2Outras Estratégias de Seleção....10
1.3.3Outras Estratégias de codificação....11
1.3.4Outras Estratégias de Formação de Populações....14
1.4Exemplo: Algoritmo genético para resolver equações de 1º grau....15
2Simulated Annealing....23
2.1Passos do Algoritmo....23
2.2Exemplo: Um SA para resolver equações de 1º Grau....25
1.5Exercícios....30
Apêndice A – Código Fonte do Algoritmo Genético para Resolução de Equações de 1º Grau....31
Apêndice B – Código Fonte do SA para Resolução de Equações de 1º Grau....37

Índice de Figuras

Figura 1: Fluxograma geral de um algoritmo genético.....5
Figura 2: Cruzamento entre dois cromossomos.....7
Figura 3: Mutação de genes em um cromossomo.....8
Figura 4: Crossover de dois pontos.....9
Figura 5: Codificação de soluções usando números reais.....12
Figura 6: Diagrama de dispersão do número de gerações.....21
Figura 7: Diagrama de dispersão dos tempos de execução.....22
Figura 8: Diagrama de dispersão de tempos de execução do SA.....28
Figura 9: Diagrama de dispersão dos resultados do SA.....28

Índice de tabelas

Tabela 1: Resultados de 15 populações para o problema da equação de 1º grau.....20
Tabela 2: Resultados de 15 execuções do SA para o problema da equação de 1º grau.....27

Métodos Heurísticos

Métodos heurísticos[1] são algoritmos exploratórios que buscam resolver problemas. Geralmente não envolvem a implementação computacional de um conhecimento especializado (por exemplo, um método heurístico, para resolver uma equação de segundo grau, não usaria, necessariamente, a fórmula de Báscara, mas buscaria, por outros métodos, uma solução que atendesse à equação). Por este motivo, muitas vezes, esses métodos são classificados como “busca cega”.

Uma solução ótima de um problema nem sempre é o alvo dos métodos heurísticos, uma vez que, tendo como ponto de partida uma solução viável, baseiam-se em sucessivas aproximações direcionadas a um ponto ótimo. Logo, estes métodos costumam encontrar as melhores soluções possíveis para problemas, e não soluções exatas, perfeitas, definitivas.

Esta subjetividade, ou falta de precisão dos métodos heurísticos, não se trata de uma deficiência, mas uma particularidade análoga à inteligência humana. Muitas vezes, no cotidiano, resolvemos diversos problemas sem conhecê-los com precisão. Alguns exemplos: ao estacionarmos um veículo, não nos preocupamos com o tamanho exato do mesmo e das vagas disponíveis; ao adoçarmos uma bebida, pouco conhecemos as propriedades do soluto e do solvente; ao tentarmos mover ou levantar um objeto, pouco nos preocupamos com leis da física que possam auxiliar ou comprometer a ação. Nestas e em diversas outras situações, a melhor solução imediata é encontrada e adotada, em detrimento de soluções comprovadamente ótimas e precisas.

Em (Colin, 2007), há uma ótima analogia entre os métodos heurísticos e o problema de localizar, empiricamente, o ponto mais alto da Terra. Para resolver este

1 Heurística é um conjunto de regras e métodos que conduzem à descoberta, à invenção e à resolução de problemas.

problema, partiria-se de um ponto viável, ou seja, de qualquer lugar na superfície terrestre, em busca das montanhas mais altas. Neste processo, várias montanhas seriam escaladas e suas alturas comparadas. O ponto mais alto iria progressivamente aumentando com as novas descobertas (os chamados ótimos locais). Até que em determinado momento as buscas se dessem por encerradas por algum motivo (a não descoberta de pontos mais altos por um longo período, falta de segurança, restrições de tempo ou financeiras) e o ponto mais alto fosse definido, mesmo sem uma comprovação científica, mesmo que outro mais alto possa ainda estar oculto.

Assim são os métodos heurísticos. Uma busca contínua e empírica, com vários ótimos locais, cujo resultado é o melhor que se pode encontrar sob determinadas condições.

Estudaremos neste capítulo os métodos: algoritmos genéticos e *simulated annealing.*

1 Algoritmos Genéticos

Este método é baseado na genética (Colin, 2007), como o próprio nome já diz, e na seleção natural. É uma simulação computacional iterativa que faz analogia a um processo evolutivo de várias gerações de uma população (Santos, 2002), onde cada indivíduo é uma representação abstrata de uma solução do problema, a seleção natural é um critério de escolha das melhores soluções e eliminação das ruins, o cruzamento e a mutação são meios para a obtenção de novas soluções.

Os algoritmos genéticos têm características bastante peculiares em relação a outros métodos de busca:

- São baseados em um conjunto de soluções possíveis;

- Não envolvem modelagem do problema (a modelagem é restrita às soluções);
- O algoritmo apresenta como resultado uma população de soluções (classificadas qualitativamente pela seleção natural) e não apenas uma;
- Trata-se de um método probabilístico e não determinístico. Em outras palavras, uma mesma população dificilmente apresentará os mesmos resultados para um mesmo problema.

1.1 Terminologia

Os algoritmos genéticos tomam emprestado vários termos da genética, com a qual mantêm conceitos análogos. Os termos principais são:

- População: conjunto de cromossomos ou soluções;
- Cromossomo: conjunto de genes. Cada cromossomo representa uma solução do problema. Muitas vezes é tomado como sinônimo de indivíduo;
- Gene: menor unidade de informação em um cromossomo. Cada gene representa uma variável da solução do problema;
- *Locus*: posição de um gene em um cromossomo. Alguns genes podem ter seu *locus* alterado em processos de cruzamento ou mutação;
- Cruzamento (ou *crossover*): processo de reprodução sexuada em que há combinação de genes dos cromossomos originando um ou mais descendentes. O cruzamento é o principal responsável pela variabilidade genética;

- Mutação: anomalias que causam a alteração aleatória de genes, seja na sua localização, seja no seu conteúdo;
- Seleção natural: processo que elimina os indivíduos menos adaptados (ou aptos) em relação à cada geração da população;
- Geração: iteração do algoritmo genético;
- Aptidão (ou *fitness*): indicador qualitativo de um indivíduo. O grau de aptidão de um indivíduo é obtido a partir de uma função objetivo;
- Função objetivo: função matemática que avalia as soluções (indivíduos) em relação ao problema.

1.2 Passos do algoritmo

Embora haja uma grande variedade e implementações, a estrutura geral de um algoritmo genético é basicamente a mesma. Dado que a representação de soluções dos problemas devem ser modeladas em forma de cromossomos, temos os seguintes passos:

Criar população inicial: que pode ser gerada aleatoriamente com *m* cromossomos;
Nova Geração: criar uma nova geração (de tamanho fixo (*m)* ou variável)

1 – Selecionar *n* cromossomos para reprodução;
2 – Realizar cruzamentos de forma aleatória;
3 – Permitir que mutações aleatórias ocorram em alguns dos cromossomos gerados;
4 – Selecionar os cromossomos melhores adaptados para a próxima geração e eliminar os menos adaptados (análogo à seleção natural);
5 – Substituir os indivíduos eliminados por novos indivíduos;

Critério de Parada: Caso a solução atenda a um critério de parada, ou caso seja detectada convergência da população, PARE. Caso contrário vá para o passo NOVA GERAÇÃO.

Para melhor compreensão, estes passos podem ser visualizados na Figura 1.

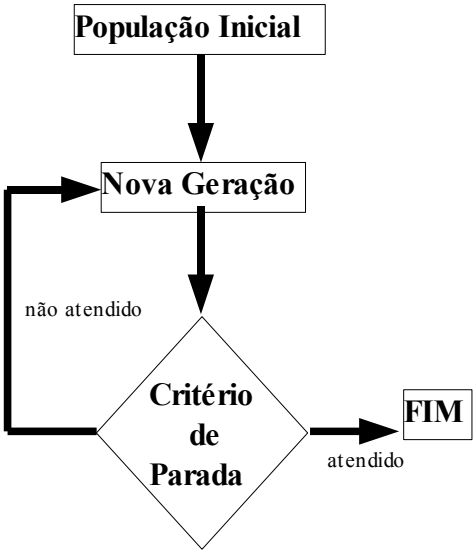

Figura 1: Fluxograma geral de um algoritmo genético

As maiores variações que se pode encontrar em algoritmos genéticos envolvem, normalmente, a seleção de indivíduos, cruzamento e mutação, e critérios de parada. Estas variações são baseadas em particularidades dos problemas a serem resolvidos.

Nas próximas subseções veremos em mais detalhes os métodos de seleção, cruzamento e mutação.

1.2.1 Seleção

Os métodos de seleção podem ser usados tanto na escolha de quais indivíduos serão progenitores, quanto na escolha dos melhores adaptados para passar à próxima geração (Santos, 2002). A seleção é baseada em um método de avaliação de aptidão de indivíduos.

Quanto mais apto um indivíduo, maior sua probabilidade de realizar cruzamentos (uma vez que indivíduos bem adaptados tendem a criar descendentes também aptos) e

melhor é sua resposta em relação ao problema (se um algoritmo genético chegar a uma solução ótima, o cromossomo que representa esta solução terá aptidão máxima).

A aptidão de um indivíduo pode ser obtida pela equação:

$$Aptidão_i = \frac{f_i}{\sum_{i}^{m} f_i}$$

Onde *i* é o índice que identifica um indivíduo na população, *m* é o tamanho da população e f_i uma função objetivo relacionada ao problema. O resultado desta equação consiste em uma probabilidade, logo um indivíduo com aptidão alta tem maior probabilidade de ser escolhido.

1.2.2 Cruzamento

Os métodos de cruzamento são responsáveis pela reprodução de cromossomos e a mistura de genes, o que garante a diversidade e a constante evolução populacional. Basicamente, este mecanismo consiste em selecionar os pontos de cruzamento nos progenitores, separar os cromossomos, e trocar as partes destes cromossomos (Goldbarg e Luna, 2000). Este mecanismo também é chamado de *crossover* (Santos, 2002).

A Figura 2 ilustra o um cruzamento de dois cromossomos, cujos genes, a título de exemplo, consistem em números binários. Note que foi selecionado apenas um ponto de cruzamento, entre o quinto e sexto gene, ou *locus*, e os descendentes receberam as cargas genéticas trocadas com base neste ponto.

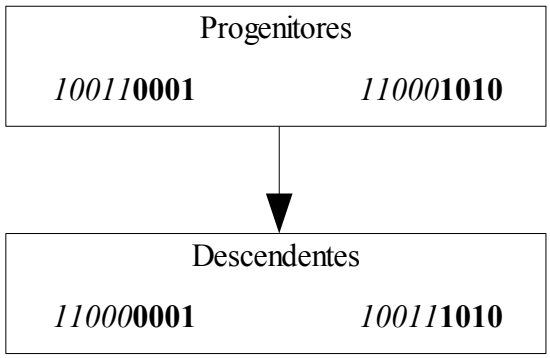

Figura 2: Cruzamento entre dois cromossomos

Este mecanismo pode sofrer uma série de variações. Uma delas é a definição de vários pontos de cruzamento, podendo gerar um número crescente de descendentes por cruzamento. Muitas vezes, ao processo de cruzamento é associado o método de mutação.

1.2.3 Mutação

A mutação consiste em selecionar um ou mais *locus* cujos genes devem ser alterados. Geralmente ocorre na criação do cromossomo, ou seja, durante ou logo após o cruzamento. A mutação, de forma análoga à biologia evolutiva, é um importante fenômeno para a diversidade e evolução, podendo ser benéfica, tornando o indivíduo mais apto, ou maléficas, condenando o indivíduo a não sobreviver à seleção natural. Portanto, ela pode desencadear o surgimento de soluções melhores, bem como soluções inferiores ou inviáveis.

A Figura 3 ilustra duas diferentes mutações em um mesmo cromossomo. Note que em negrito estão nos *locus* escolhidos para a troca de genes (o terceiro gene passou para o lugar do oitavo gene, e este passou a ocupar o terceiro *locus*). Já o *locus* sublinhado teve seu gene alterado de 0 para 1.

Figura 3: Mutação de genes em um cromossomo.

Apesar de sua importância para a diversidade e evolução, a mutação excessiva em uma população pode acarretar várias soluções anômalas, impedindo a evolução da população.

1.3 Variações das Etapas dos Algoritmos Genéticos

Baseado nos conceitos apresentados, passaremos a analisar variantes das etapas dos algoritmos genéticos. Inicialmente trataremos das variações dos operadores cruzamento. A seguir serão tratadas outras possibilidades de seleção e codificação. Por fim, serão analisadas algumas alternativas para a formação de populações.

1.3.1 Outras Estratégias de Crossover

Dentre as várias alternativas existentes, nesta seção analisaremos o *crossover* de dois pontos e o *crossover* uniforme.

Crossover de dois pontos

No *crossover* de dois pontos, apenas o material genético entre os *locus* escolhidos são trocados. Na Figura 4, os *locus* escolhidos foram o terceiro e o sexto. Pode-se notar

que os genes à esquerda e à direita do intervalo escolhido permanecem inalterados, ao contrário dos genes dentro do intervalo.

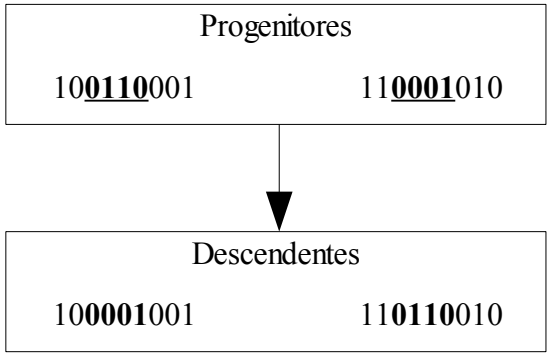

Figura 4: Crossover de dois pontos

Esta variação do *crossover* tende a tornar o algoritmo genético mais eficiente, uma vez que contribui para a variabilidade genética (Linden, 2006). Este método abrange também o *crossover* de um ponto, visto anteriormente, caso o primeiro ou último gene sejam escolhidos.

Crossover uniforme

O *crossover* uniforme consiste em um sorteio binário para cada gene, definindo como os genes de cada progenitor serão distribuídos entre os descendentes. Por exemplo, caso seja sorteado 1 para um determinado gene, o primeiro descendente recebe o gene do primeiro progenitor e o segundo descendente recebe o gene do segundo progenitor. Caso seja sorteado 0, o processo se inverte. Este sorteio é realizado para cada um dos genes dos cromossomos.

Esta estratégia de *crossover* tende a prover maior diversidade, porém tem um

custo de processamento mais alto, uma vez que há um sorteio para cada gene de cada indivíduo.

1.3.2 Outras Estratégias de Seleção

As estratégias de seleção podem acelerar ou retardar tanto a busca pela solução ótima, quanto a convergência, uma vez que definem os cromossomos que irão gerar e participar da próxima geração. Dada a importância da seleção analisaremos algumas variações.

Seleção por Torneio

Neste método, há um torneio entre indivíduos selecionados aleatoriamente, sem qualquer favorecimento aos melhores adaptados (Linden, 2006). Uma vez selecionados os indivíduos, os mais aptos vencerão o torneio. Devem ser definidos previamente o número de indivíduos a participarem de cada torneio e quantos torneios serão realizados.

A vantagem deste método é a impossibilidade de alguns cromossomos dominarem a população, o que levaria a uma rápida convergência.

Seleção por Vizinhança

Neste método, também chamado de seleção local, são definidas vizinhanças aleatórias, ou seja, intervalos aleatórios de indivíduos. Os indivíduos destas vizinhanças

poderão realizar *crossover* entre si. Caso haja interseção entre vizinhanças (indivíduos em mais de uma vizinhança), os indivíduos destas poderão interagir entre si. As interseções são bastante desejáveis, uma vez que impede que o *crossover* seja segmentado em pedaços pequenos da população, o que acarretaria em baixa variabilidade.

1.3.3 Outras Estratégias de codificação

O método de codificação de soluções apresentado foi a representação binária. Porém, esta representação apresenta algumas limitações(Linden, 2006):

- Variáveis contínuas e de alta precisão podem resultar em cromossomos excessivamente grandes;
- O número de soluções codificadas é sempre uma potência de dois: 2^k, onde k é o número de genes de um cromossomo. Caso a variável representada descreva um número finito de estados que não seja uma potência de dois, várias soluções possíveis serão inválidas[2];

Portanto podemos fazer uso de outras estratégias de codificação.

2 Por exemplo: uma variável pode ter apenas cinco estados; neste caso serão necessários dois genes, pois $2^3 = 8$. Logo, três soluções serão inválidas, o que exige um certo tratamento, tornando o algoritmo mais complexo e menos eficiente.

Representação numérica

Em muitos problemas pode ser mais conveniente o uso cromossomos com números reais ao invés de binários. A representação numérica não traz limitações quanto a precisão e permite que os cromossomos tenham tamanho mínimo.

Utilizando representação numérica, o cromossomo do problema da equação de 2º grau teria apenas dois genes, um para cada solução. A Figura 5 apresenta algumas soluções codificadas em números reais.

1	5
20	3
8,5	2
0	-15
2,5	3
4	1

Figura 5: Codificação de soluções usando números reais

As vantagens desta estratégia de codificação são: a possibilidade de representar casas decimais, números negativos e o tamanho reduzido dos cromossomos. Entretanto os operadores genéticos devem ser adaptados:

- O *crossover* será mais simples, permitindo apenas a permuta dos pares de genes de cada cromossomo;

- A mutação envolverá a inversão dos genes ou a alteração do valor contido neles;

A representação numérica é flexível e abrangente, podendo se utilizada em vários tipos de problemas que envolvam variáveis quantitativas. Porém, quando se trabalha com variáveis categóricas, um outro tipo de representação é necessária.

Representação categórica

Na representação categórica, são codificados conjuntos valores predefinidos. São exemplos de variáveis categóricas:

- Direções: esquerda, direita, frente e trás; ou norte, sul, leste e oeste;
- Bases de DNA: A, C, G e T;
- Substâncias químicas: Na, Cl, O, N, H e outros.

Esta representação pode codificar soluções de problemas como:

- Caminho para sair de um labirinto;
- Busca de um genoma;
- Identificação ou composição de uma substância química;

O operador genético que necessitaria de uma maior adaptação seria a mutação, uma vez que, ao alterar o conteúdo de um gene, há um conjunto limitado de

possibilidades para mutação.

1.3.4 Outras Estratégias de Formação de Populações

Visando tirar o melhor proveito das qualidades de cada geração, pode-se adotar diferentes alternativas para se compor uma população.

O tamanho é um importante critério na formação e evolução de uma população, pois uma população muito pequena dificilmente alcançará uma grande variedade genética. Já uma população muito grande (embora tenha maior diversidade) tornará a execução bastante pesada, afetando a eficiência do algoritmo genético. Infelizmente não existe um tamanho padrão indicado para todos os tipos de problema. Modelagens com pequenos cromossomos podem ter grandes populações sem perda de eficiência. Já modelagens com grandes cromossomos devem ter populações de tamanho limitado devido ao custo de processamento. Portanto, deve-se obter empiricamente o tamanho ideal para cada problema, podendo-se adotar um valor inicial para testes, valor este baseado no tamanho do cromossomo. Por exemplo: 20 * Tamanho do Cromossomo.

Outra alternativa é adotar populações de tamanho variável. Isto pode ser feito associando uma idade a cada indivíduo. A medida que a idade vai aumentando, a cada geração, aumenta a probabilidade de “morte” dos indivíduos. Portanto, o tamanho da população varia com a “taxa de natalidade” (baseada nos critérios de seleção e *crossover*) e com a “taxa de mortalidade”.

Além das definições de tamanho, pode-se adotar a técnica de elitismo. Nesta técnica, normalmente integrada ao processo de seleção, determinado número de melhores indivíduos deve passar a próxima geração, garantindo que boas soluções

permaneçam na população e possam gerar soluções melhores nas próximas gerações.

1.4 *Exemplo: Algoritmo genético para resolver equações de 1º grau.*

Uma equação de 1º grau completa é dada no seguinte formato:

$$bx - c = 0$$

Uma equação do 1º grau pode ser facilmente resolvida por operações matemáticas simples. Mas vamos desenvolver um exemplo didático de algoritmo genético para introduzir o leitor à implementação deste método.

Equação utilizada

Para este exemplo foi usada a equação abaixo, cuja cuja solução é 5.

$$2x - 10 = 0$$

Modelagem da solução

As soluções serão representadas por números binários de seis bits. Logo, cada cromossomo será constituído por seis genes, cujos valores podem ser 1 ou 0. Estes números binários são convertidos para números inteiros e posteriormente para números

reais. Apesar da solução desta equação ser inteira, é interessante trabalhar com números reais, pois vários problemas trabalham com variáveis contínuas.

Criação da população inicial

A população inicial foi criada com o uso funções geradoras de números aleatórios. Sendo cada gene de cada indivíduo definido aleatoriamente, é bastante provável que haja uma significativa variabilidade populacional na primeira geração. O tamanho da população foi fixado em cinquenta indivíduos.

Seleção

A seleção foi baseada na função objetivo:

$$f_i = \frac{1}{2^{2x-10}}$$

Esta função será aplicada a cada cromossomo. Quanto mais próximo a solução representada estiver de uma da solução ótima, mais próximo de 1 será o resultado desta função. Portanto, quando um cromossomo representar uma das soluções ótimas, o resultado da função será exatamente 1.

A aptidão de cada cromossomo será obtida a partir da fórmula geral:

$$Aptidão_i = \frac{f_i}{\sum_i^m f_i}$$

Evidentemente, quanto mais próximo das soluções ótimas, maior será a aptidão. Há duas situações em que um indivíduo pode ser escolhido*:*

- A primeira é determinística: O indivíduo cuja aptidão for maior que a probabilidade de seleção (parâmetro pré-definido que se tornará cada vez mais restritivo a cada geração, “forçando” a evolução da população) será selecionado;
- A segunda é probabilística: Para o indivíduo de aptidão menor que a probabilidade de seleção, será gerado um número aleatório. Se este número for maior que a aptidão do indivíduo, ele será selecionado. Este critério tende a selecionar indivíduos menos adaptados (uma vez que é menor a probabilidade de um número aleatório ser maior que a aptidão de indivíduos bem adaptados), visando garantir a diversidade genética e evitar a convergência da população.

Cruzamento

Para cada cruzamento são escolhidos, aleatoriamente, dois indivíduos já previamente selecionados. Em seguida, são definidos, também aleatoriamente, dois *locus* para então as cargas genéticas serem trocadas gerando dois novos indivíduos, conforme exibido na Figura 4.

Mutação

Neste exemplo, foi definida a taxa, ou porcentagem, de mutação na população, ou seja, um número fixo de mutações irá ocorrer a cada geração. Com base nesta taxa é dada a probabilidade de um indivíduo sofrer ou não mutação. Por exemplo, uma taxa de 10% indica que cada indivíduo possui probabilidade 0,1 de sofrer uma mutação. Já o *locus* de mutação é definido aleatoriamente e os bits nestes locais são invertidos, conforme exibido na Figura 3.

Em boa parte das publicações, a mutação é totalmente aleatória, tando quanto à taxa de ocorrência a cada geração, quanto à probabilidade individual. Porém, a parametrização deste operador genético permite um maior controle sobre a diversidade e evolução da população, o que pode ser desejável quando se está adaptando um algoritmo genético para diferentes problemas, ou quando se busca estabelecer parâmetros otimizados de busca.

Substituição

Os indivíduos não selecionados para o cruzamento são substituídos pelos novos cromossomos. Desta forma, a população mantém tamanho constante e permite que indivíduos com alta aptidão sobrevivam a várias gerações[3].

Critério de parada

3 A aptidão é relativa, ou seja, ela varia de acordo com a evolução de uma população. Logo, um indivíduo bem adaptado em uma determinada geração, pode vir a ter uma baixa aptidão em gerações futuras.

O critério de parada é baseado em soluções quase ótimas. Quando uma solução se mantém como a melhor durante dez gerações o algoritmo é encerrado. Este critério não garante solução ótima, tampouco garante que um ótimo local seja encontrado. Como ocorre em qualquer implementação de algoritmos genéticos, há risco de a população convergir para um ótimo local, o que pode ser evitado com a variabilidade populacional.

Performance do Algoritmo

Foram criadas quinze populações, ou seja, o algoritmo genético foi executado 15 vezes para coleta e análise dos dados exibidos na Tabela 1.

Tabela 1: Resultados de 15 populações para o problema da equação de 1º grau

População	Número de Gerações	Tempo de Execução (s)
1	25	11,94
2	21	9,82
3	17	7,98
4	19	8,78
5	31	14,33
6	19	8,65
7	29	13,01
8	29	13,27
9	31	14,61
10	16	7,22
11	21	10,1
12	21	9,69
13	30	13,78
14	16	7,4
15	13	6,17
Máximo	**31**	**14,61**
Mínimo	**13**	**7,22**
Média	**22,53**	**10,45**
Coeficiente de Variação	**0,27**	**0,27**

Todos as populações encontraram uma solução ótima ou quase ótima, havendo aproximação nestes casos apenas na terceira casa decimal. Houve variação apenas no

número de gerações e tempo de execução de cada população. Em outras palavras, todas as populações foram eficazes, porém nem todas foram eficientes.

Em relação ao número de gerações, houve valores extremos de 13 e 31, com média de aproximadamente 22 gerações. Já os tempos de execução variaram entre os extremos 7,22 e 14,61 segundos. A média de tempo foi de 10,45 segundos.

Os diagramas de dispersão destes dados são exibidos na Figura 6 e Figura 7. Tanto a análise destes diagramas, quanto a análise dos coeficientes de variação(Vieira, 1999) da Tabela 1, mostram a mesma dispersão do tempo de execução e número de gerações[4]. Entretanto é bastante óbvio que o número de gerações e o tempo de execução são variáveis com forte correlação[5].

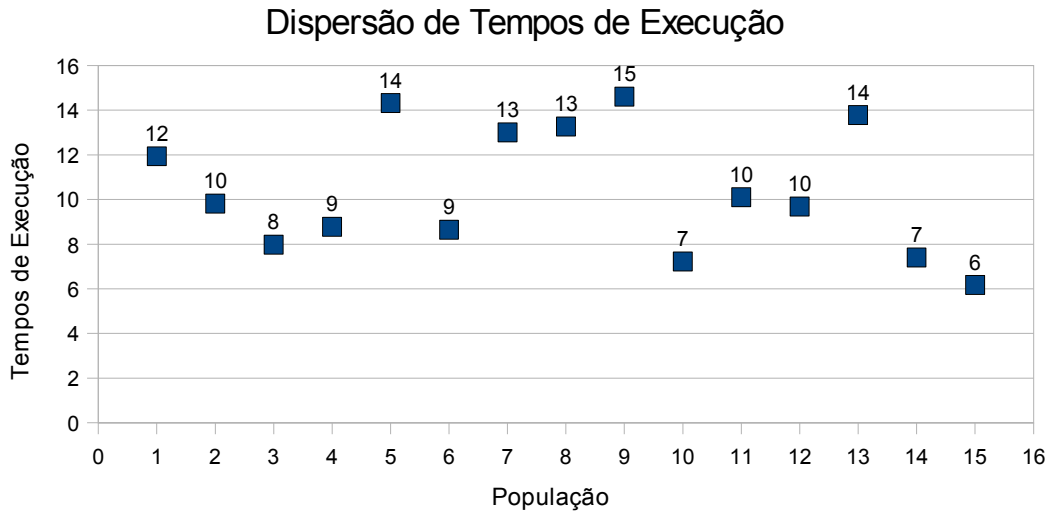

Figura 6: Diagrama de dispersão do número de gerações

4 Ao contrário do número de gerações, os tempos de execução podem variar de acordo com a capacidade de processamento dos recursos computacionais utilizados. Quanto maior o poder de processamento, menor será a variação dos tempos de execução. Este algoritmo genético foi executado em um Pentium 4 de 3GHz e 512MB de RAM.

5 A correlação destas variáveis, para o recurso computacional utilizado, foi 1, o que indica uma forte correlação.

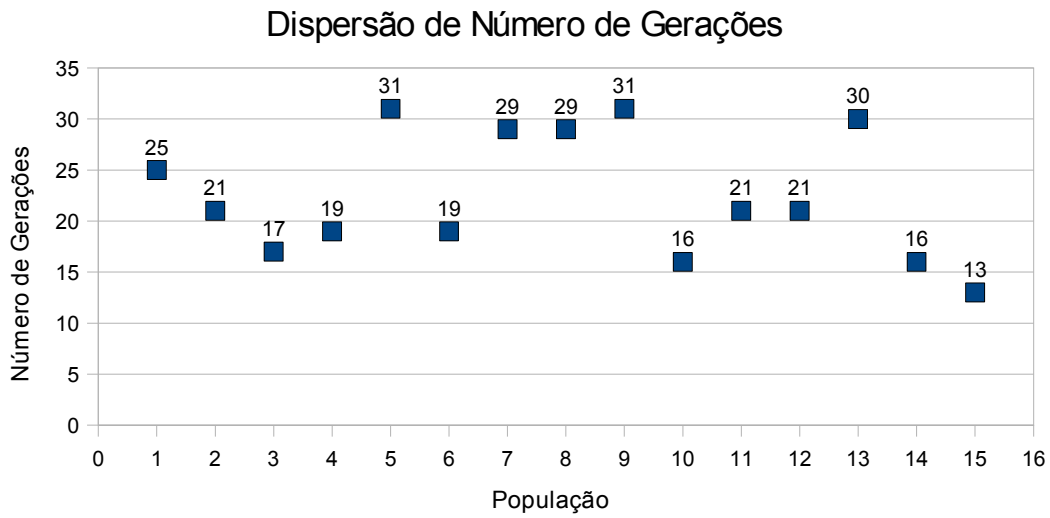

Figura 7: Diagrama de dispersão dos tempos de execução

O código fonte (em Octave®) deste algoritmo genético está disponível no Apêndice A.

2 Simulated Annealing

A metalurgia utiliza um processo de tratamento térmico visando alterar a estrutura cristalina de metais conferindo-lhes características mecânicas e estruturais desejadas. Este processo, chamado de recozimento, consiste em aquecer continuamente metais até determinada temperatura, e, posteriormente, resfriá-los em um forno com resfriamento controlado. Diferentes velocidades de resfriamento levam a diferentes propriedades nos metais. Um resfriamento muito rápido acarreta em imperfeições nos cristais metálicos. Já um resfriamento muito lento leva à formação de cristais muito grandes.

Baseado neste processo, foi criado o método heurístico chamado *simulated annealing* (SA), ou recozimento simulado, em português. Neste método, parte-se de uma solução viável de um problema e passa-se a aceitar soluções vizinhas. A princípio, nas altas temperaturas (analogamente falando), há grande probabilidade de qualquer solução vizinha ser aceita. Mas, a medida que ocorre o resfriamento, há maior probabilidade de soluções melhores serem aceitas.

Como vantagens do SA pode-se citar tanto a sua capacidade de resolver problemas de diversos níveis de complexidade em várias áreas específicas, quanto a sua relativa previsibilidade e simplicidade, uma vez que trabalha com poucos parâmetros e envolve operações matemáticas e computacionais simples.

2.1 Passos do Algoritmo

Antes de iniciar um algoritmo de SA, é necessário definir os parâmetros:

- *t*: temperatura inicial;
- *r*: fator de resfriamento, que determina a velocidade de "resfriamento" do algoritmo. O fator de resfriamento deve estar no intervalo: 0<r<1;
- *k*: número de soluções vizinhas a serem testadas a cada nível de temperatura;

Além destes parâmetros, deve ser definido o critério de parada, podendo se tratar de um número máximo de iterações, uma temperatura mínima ou uma solução ótima ou quase ótima.

Os passos do SA são dados a seguir.

Criar conjunto de soluções iniciais

Inicialização dos parâmetros: *t, r e k*

Escolher uma solução inicial: *x*

Iteração:

Repita de 1 até k

1 – Escolher aleatoriamente uma solução vizinha: x^*

2 – Comparar o custo da solução atual e da nova solução:

$$\Delta = custo(x^*) - custo(x)$$

3 – Se $\Delta \leq 0$ (função objetivo diminui)

então $x = x^*$

senão se *probabilidade* $>$ $exp(-\Delta/t)$

então $x = x^*$

Reduza a temperatura: $t = t^* r$

Critério de Parada: Caso a solução atenda a um critério de parada, PARE. Caso contrário realize uma nova iteração.

O algoritmo acima exposto é aplicavel a um problema de minimização, uma vez que Δ será menor ou igual a zero se o custo da nova solução (x^*) for menor ou igual ao custo da solução atual (x). Mesmo que a nova solução não minimize o custo, ela ainda assim pode ser aceita. Para tanto, deve-se gerar um número aleatório (ou seja uma probabilidade), e, se este for maior que $exp(-\Delta/t)$, a solução será aceita. Note que à medida que a temperatura t diminui, o valor de $exp(-\Delta/t)$ aumenta, consequentemente haverá menor chance de se gerar um número aleatório maior que este valor e, portanto, menor probabilidade de uma solução ruim ser aceita. É exatamente esta a essência do SA.

Caso o valor de r seja muito pequeno, haverá um resfriamento rápido, fazendo com que o algoritmo se limite a uma busca local (Colin, 2007). Entretanto, um r muito grande (próximo de 1) faz com que o algoritmo possa gastar muitas iterações com soluções ruins.

Para um problema de maximização, este algoritmo pode ser facilmente adaptado. Basta considerar aceitáveis as soluções com $\Delta \geq 0$, ou com $probabilidade < exp(\Delta/t)$.

2.2 Exemplo: Um SA para resolver equações de 1º Grau

Uma equação de 1º grau completa é dada no seguinte formato:

$$bx - c = 0$$

Uma equação do 1º grau pode ser facilmente resolvida por operações matemáticas simples. Porém, vamos desenvolver um exemplo didático de SA para resolver uma equação de 1º grau , afim de introduzir o leitor à implementação deste algoritmo.

Equação utilizada

Para este exemplo também foi usada a equação:

$$2x - 10 = 0$$

A solução desta equação é x=5, e sua modelagem para SA não envolve nenhuma adaptação, ao contrário da implementação em Algoritmos Genéticos.

Criação das soluções iniciais

As soluções iniciais foram criadas com o uso funções geradoras de números aleatórios em torno de uma solução inicial. O número de soluções foi fixado em dez.

Critério de parada

Como critério de parada foi definida a temperatura de 10^{-6}. Temperaturas mais baixas demonstraram pouco contribuir com um melhor resultado.

Performance do Algoritmo

O algoritmo foi executado 15 vezes para coleta e análise dos dados exibidos na Tabela 2.

Tabela 2: Resultados de 15 execuções do SA para o problema da equação de 1º grau

Execução	Resultado	Tempo de Execução (s)
1	4,99	0,11
2	5,01	0,12
3	5,02	0,11
4	5,00	0,12
5	5,06	0,12
6	4,97	0,13
7	5,02	0,12
8	4,97	0,12
9	5,02	0,13
10	4,98	0,12
11	5,01	0,12
12	4,97	0,12
13	5,02	0,12
14	5,17	0,12
15	5,04	0,11
Máximo	**5,17**	**0,13**
Mínimo	**4,97**	**0,11**
Média	**5,02**	**0,12**
Coeficiente de Variação	**0,01**	**0,05**

Esta implementação de SA se mostrou tanto eficiente quanto eficaz (embora a implementação em Algoritmos Genéticos tenha tido resultados mais precisos), uma vez que as execuções exigiram pouco tempo de processamento e retornaram valores

bastante próximos do resultado ótimo. São estas características, nas suas devidas proporções, que tornam o SA atraente para problemas mais complexos.

Os diagramas de dispersão destes dados são exibidos na Figura 8 e Figura 9. Ficam evidenciados nestes diagramas a baixa dispersão das execuções do SA. Esta é outra característica bastante desejável do SA, a previsibilidade.

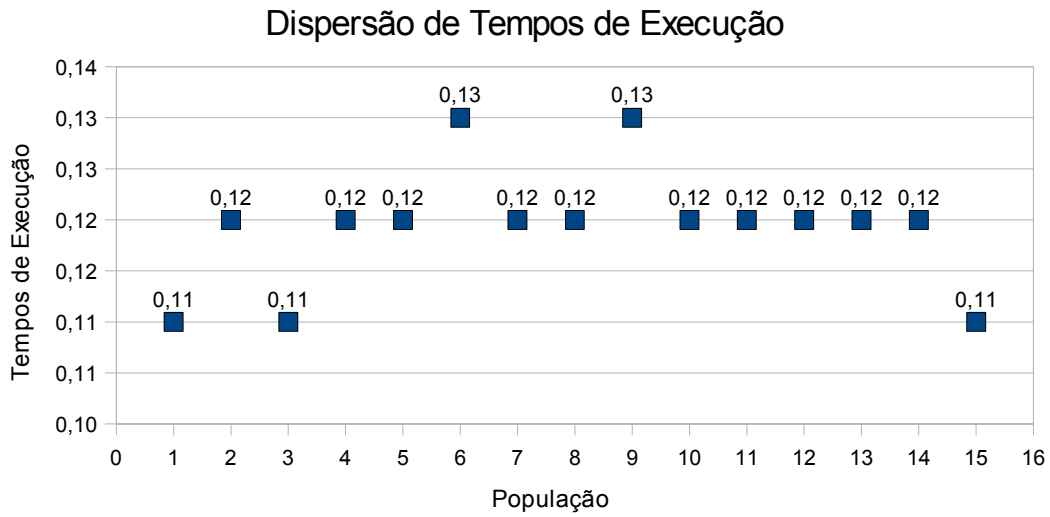

Figura 8: Diagrama de dispersão de tempos de execução do SA

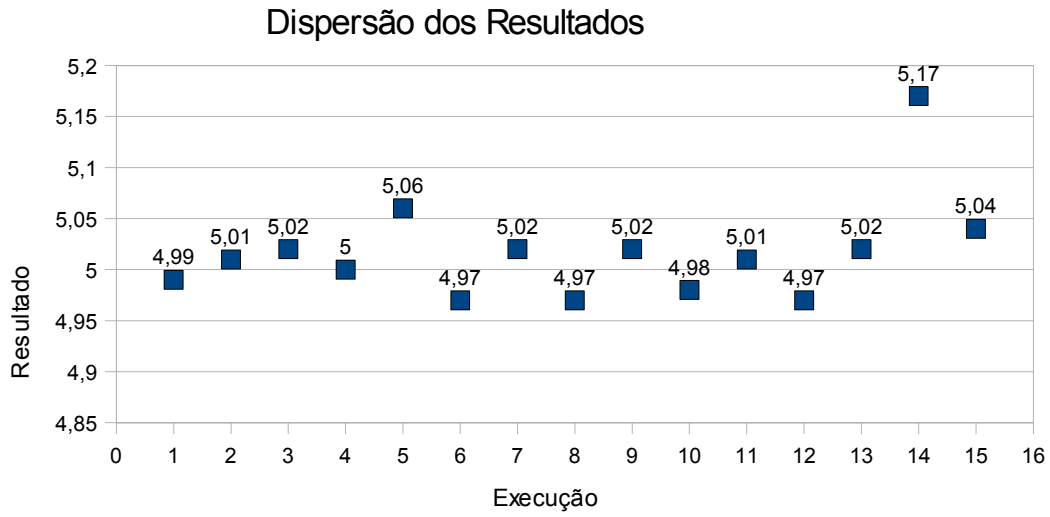

Figura 9: Diagrama de dispersão dos resultados do SA

O código fonte (em Octave®) deste exemplo está disponível no Apêndice B.

1.5 Exercícios

1. Utilizando o exemplo estudado na seção 1.4, experimente variar os parâmetros tamanho da população, tamanho do cromossomo, probabilidade inicial de seleção e probabilidade de mutação, verifique como a eficácia e eficiência do algoritmo são afetadas.
2. Implemente a estratégia de *crossover* uniforme no algoritmo da seção 1.4. Verifique os ganhos ou perdas de performance obtidos.
3. Implemente as estratégias de seleção por torneio e por vizinhança no algoritmo da seção 1.4. Verifique, para cada uma destas estratégias, os ganhos ou perdas de performance obtidos.
4. Modifique o algoritmo da seção 1.4 para que o tamanho da população seja variável. Verifique os ganhos ou perdas de performance obtidos.
5. Altere os parâmetros do algoritmo SA da seção 2.2 e verifique as variações dos resultados e tempos de execução.

Apêndice A – Código Fonte do Algoritmo Genético para Resolução de Equações de 1º Grau

```
clc

tempo_inicial=clock();
tempo_inicial=tempo_inicial(6);
%%%%%parametros%%%%%%

sup=100;
inf=0;

%tamanho da população
m=50
%tamanho do cromossomo
n=4;
%probabilidade inicial de selecao
ps=0.1;

%probabilidade de mutação
pm=0.5;


%inicia vetor de soluções
solucoes=[];

%%%%%%%%%%criação da primeira geração

%%%%%primeiro cromossomo é formado por uma sequencia de 1
cromossomos=ones(1,n);


melhor=-1;
cont_melhor=0;
x_melhor=[];

%inicia contador
i=1;

b=2;
c=-10;

%repetir até atingir tamanho da população
while (i<=m-1)

    %insere primeiro gene
    if rand>0.5
        novo_cromossomo=1;
    else
        novo_cromossomo=0;
    endif
    %cria um indivíduo (repete até que seja atingido o tamanho do cromossomo)
    for j=1:n-1
        if rand>0.5
            novo_cromossomo=[novo_cromossomo 1];
        else
```

```
                novo_cromossomo=[novo_cromossomo 0];
            endif
        endfor

        cromossomos=[cromossomos; novo_cromossomo];

        i++;
endwhile

geracoes=1;

% início do processo iterativo
while (1)


%%%%%%%%%criação de nova geração


%%%%%%%%%%%%%%%%%%%%%%%%%%%%        seleção

  %inicia vetor de resultados da funcao objetivo e aptidoes
  funcao=zeros(m,1);
  aptidao=zeros(m,1);

  individuos_reais=[];

  %realiza teste de aptidao para cada cromossomo
  for i=1:m
    %valor de x para cromossomo
    %cromossomos(i,:)
    %cromossomos(i,1:n)
     x=bin2dec(num2str(cromossomos(i,1:n)));

    %converte o número inteiro para real
     x=inf+((sup-inf)/2^n-1)*x    -rand;

     %armazena individuos convertidos para numeros reais
      individuos_reais=[individuos_reais; x];

     %obtem resultado da equacao
     teste=b*x+c;
     %funçãoo objetivo. Quanto mais próximo o resultado da equação estiver de zero, melhor será o
valor de x, mais próximo funcao estará de 1
    funcao(i)=1/(2^abs(teste));
    %funcao(i) = teste;
  endfor

%[individuos_reais funcao]

  %soma todo vetor funcao
  soma=sum(funcao);
  %média das funcoes
  media=soma/m;

  selecionados=[];
```

```
i=1;
num_selecionados=0;
while num_selecionados<=ps*m


  %cada individuo é escolhido de forma determinística
  if (funcao(i)>=max(funcao)-abs(media)/2)
    selecionados=[selecionados i];
    num_selecionados++;
  endif
  i++;
  if(i>m)
    break;
  endif
endwhile



 %caso nenhum indivíduo tenha sido selecionado, metade da população é escolhida
 if(size(selecionados,1)==0)
   for(i=1:m/2)
     selecionados(i)=i;
   endfor
 endif

 %a probabilidade deve ir decrescendo à medida que a população converge (no início muitos podem
ser selecionados, mas gradativamente
 %a seleção vai se tornando mais restritiva)
 ps=ps+(1/m^1);



 %%%%%%%%%%%%%%%%%%%%%%%%%%%       fim de seleção

 %%%%%%%%%%%%%%%%%%%%%%%%       critério de parada

  %verifica se há solução melhor que a atual
 if(melhor<max(funcao) )
   %nova solução passa a ser a melhor e sua posição é armazenada
   [melhor posicao]=max(funcao);
   cont_melhor=0;
   %novo melhor x é armazenado
   x_melhor=individuos_reais(posicao,:) ;
   else %caso não haja solução melhor que a atual, contador é incrementado
     cont_melhor=cont_melhor+1;
     %se uma mesmo solução for a melhor durante 10 gerações, então programa se encerra
     if((cont_melhor==10) & (melhor!=-1))
       x_melhor
       melhor
       break;

      endif
 endif


   %%%%%%%%%%%%%%%%%%%%%%%%       fim do critério de parada
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%       cruzamento


if(m==num_selecionados)
   novos_individuos=round(num_selecionados/2);
  else
  novos_individuos=round(m-num_selecionados);
 endif
if(mod(novos_individuos,2)!=0)
  round(novos_individuos++);
endif
num_selecionados;


novos_cromossomos=[];
i=1;

while i<=novos_individuos
  %seleciona individuos aleatoriamente
  individuo1=round(rand*10);
  if(individuo1>num_selecionados | individuo1==0)
    individuo1=num_selecionados;
  endif

  individuo2=round(rand*10);
  if(individuo2>num_selecionados | individuo2==0)
    individuo2=1;
  endif

  %seleciona pontos de crossover aleatoriamente
  crossover1=round(rand*10);
  while(crossover1>=n/2 | crossover1==0)
    crossover1=round(rand*10);
  endwhile

  crossover2=crossover1+n/2;

 % crossover2=round(rand*10);
  %while(crossover2>=2*n | crossover2==0 | crossover2<=crossover1)
   % crossover2=round(rand*10);
  %endwhile

  %operação de crossover de dois pontos
  novos_cromossomos=[novos_cromossomos;
cromossomos(selecionados(individuo1),1:crossover1)
cromossomos(selecionados(individuo2),(crossover1+1):crossover2)
cromossomos(selecionados(individuo1),(crossover2+1):n)];
  novos_cromossomos=[novos_cromossomos;
cromossomos(selecionados(individuo2),1:crossover1)
cromossomos(selecionados(individuo1),(crossover1+1):crossover2)
cromossomos(selecionados(individuo2),(crossover2+1):n)];



  %soma a quantidade de indivíduos criados ao índice i
  i=i+2;
```

```
endwhile


%inclusão cromossomos selecionados para próxima geração
novos_cromossomos=[novos_cromossomos ;  cromossomos(selecionados,:)];


%%%%%%%%%%%%%%%%%%%%%%%%%%%       mutação

%obtém número de mutações
num_mutacoes=m*pm;

for i=1:num_mutacoes
  %escolha de indivíduo para mutação
  individuo=round(rand*100);
  while(individuo>m | individuo==0)
    individuo=round(rand*100);
  endwhile

  %escolha de locus para mutação
  locus=round(rand*10);
  while(locus>n | locus==0)
    locus=round(rand*10);
  endwhile


  %efetua inversão do bit na posição selecionada
  novos_cromossomos(individuo,locus)=not(novos_cromossomos(individuo,locus)
);

  %escolha de locus para mutação
  locus=round(rand*10);
  while(locus>n | locus==0)
    locus=round(rand*10);
  endwhile

  %efetua inversão do bit na posição selecionada
  novos_cromossomos(individuo,locus)=not(novos_cromossomos(individuo,locus)
);

%escolha de locus para mutação
  locus=round(rand*10);
  while(locus>n | locus==0)
    locus=round(rand*10);
  endwhile

  %efetua inversão do bit na posição selecionada
  novos_cromossomos(individuo,locus)=not(novos_cromossomos(individuo,locus)
);


endfor

%geração anterior é substituída pela atual
cromossomos=novos_cromossomos;
```

```
 geracoes++;
 ps
 geracoes

 %se probabilidade de seleção for negativa, ela é resetada e a probabilidade de mutação dobrada
 if(ps<0)
   ps=0.1;
   pm=pm*2;
  endif
endwhile

%obtém tempo de execução
 tempo_final=clock();
 tempo_final=tempo_final(6);

 %exibe tempos
 tempo_inicial
 tempo_final
 tempo_final-tempo_inicial
```

Apêndice B – Código Fonte do SA para Resolução de Equações de 1º Grau

```
tempo_inicial=clock();

tempo_inicial=tempo_inicial(6);


%temperatura inicial
t=5;
%repetições
k=5;
%fator de resfriamento
r=0.8;
%número de iterações
num_iter=50;
%solução inicial
x=0;
%solução atual
x_atual=x;

%número de soluções
num_solucoes=10;

i=1;
while(t>10^-6)

    %cria vizinhança de soluções
    x_maximo=x_atual+10;
    x_minimo=x_atual-10;
    solucoes=rand(num_solucoes,1)*(x_maximo-x_minimo)+x_minimo;
    solucoes=[x;solucoes];

    for(m=1:k)


        %escolhe vizinho aleatoriamente
        vizinho=int32(rand*(num_solucoes-1)+1);

        %cálculo de custos
        custo_atual=abs(2*x_atual-10);
        x=solucoes(vizinho);
        custo=abs(2*x-10);
        delta=custo-custo_atual;

        %teste de aceitação da solução escolhida
        if(delta<=0)
            x_atual=x
        else
            probabilidade=exp(-delta/t);
            teste=rand;
```

```
                    if probabilidade>teste
                        x_atual=x
                    endif
                endif

        endfor

        %redução da temperatura
        t=r*t;
         i=i+1;
endwhile
x_atual
t
i

%obtém tempo de execução
  tempo_final=clock();
  tempo_final=tempo_final(6);

  %exibe tempos
  tempo_inicial;
  tempo_final;
  tempo_final-tempo_inicial
```

Bibliografia

COLIN, Emerson Carlos. **Pesquisa Operacional: 170 aplicações em estratégia, finanças, logística, produção, marketing e vendas**.Rio de Janeiro: LTC, 2007

SANTOS, Fabríco B. B.. Implementação Eficiente de Busca em Plataforma Paralela. In: XXII Congresso da Sociedade Brasileira de Computação, Florianópolis, 2002. Anais. Florianópolis, SBC, 2002.

GOLDBARG, M. C. LUNA, H. P . **Otimização Combinatória e Programação Linear: modelos e algoritmos**.Rio de Janeiro: Campus, 2000

LINDEN, Ricardo. **Algoritmos Genéticos**.Rio de Janeiro: Brasport, 2006

VIEIRA, Sônia. **Estatística para a Qualidade**.Rio de Janeiro: Campus, 1999