Análise de desempenho de algoritmos genéticos, Hill Climbing e Simulated Annealing

Autor

André Furlan - ensismoebius@gmail.com

O projeto

Este projeto está licenciado sob a licença **GPL versão 3** e está armazenado em um

repositório compartilhado no endereço:

https://github.com/ensismoebius/computacaoInspiradaPelaNatureza

O documento

Este documento está licenciado sob a licença Creative Commons **Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)**.

Para mais informações acesse: https://creativecommons.org/licenses/by-sa/4.0/

A estrutura do projeto

Para melhor acompanhamento deste trabalho é importante entender a estrutura de arquivos do projeto em questão:

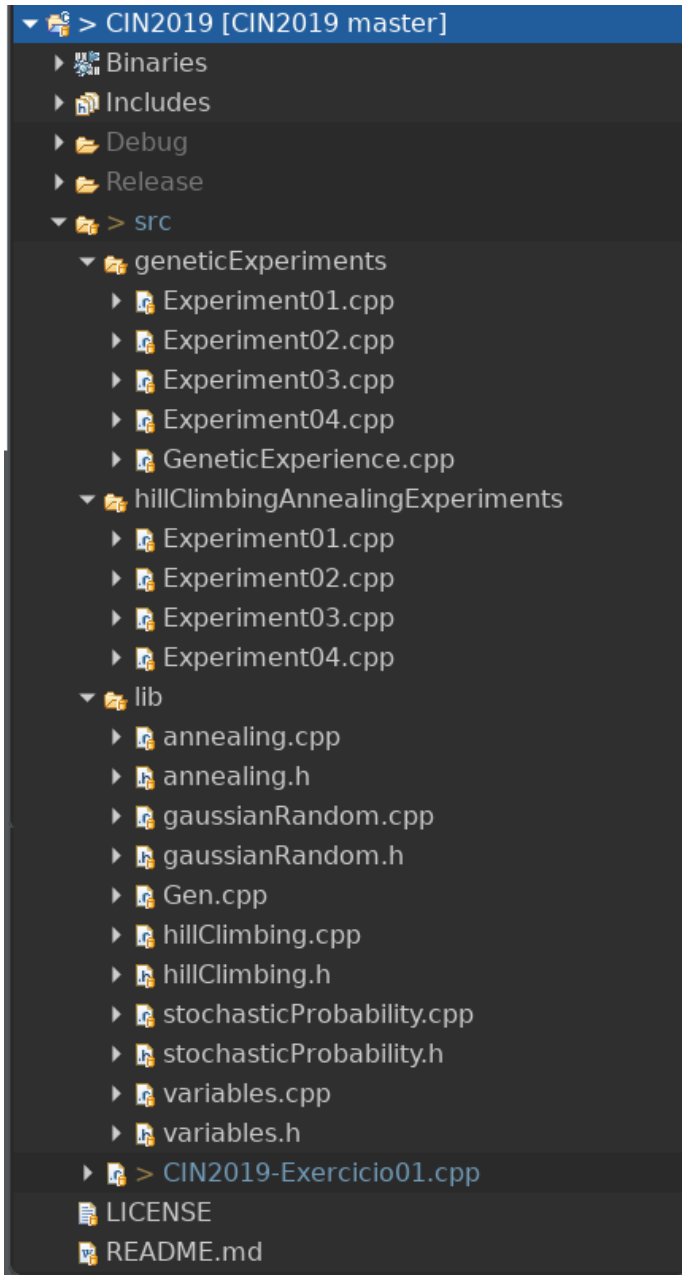

Figura 1: Estrutura do projeto

O diretório principal do projeto chama-se “src” é nele que todos os códigos residem.

O primeiro diretório que aparece dentro de “src” chama-se **“geneticExperiments”** dentro do mesmo estão as implementações de todos os experimentos com algoritmos genéticos propostos neste trabalho.

O segundo diretório que aparece chama-se **“hillClimbingAnnealingExperiments”** no mesmo estão todos os experimentos usando os algoritmos Hill Climbing e Simulated Annealing.

No terceiro diretório estão todos as bibliotecas derivadas do desenvolvimento deste trabalho cada uma delas procura agrupar e organizar as lógicas e técnicas empregadas de forma que as mesmas possam ser usadas em qualquer outro sistema, ou seja, procurou-se diminuir ao máximo o acoplamento de código.

Por fim perto do final da figura se percebe um arquivo chamado ”CIN2019-Exercicio1.cpp” dentro do qual estão as chamadas para todos os experimentos contidos dentro do projeto.

Tecnologia necessárias

Para que se possa reproduzir os experimentos aqui descritos antes de tudo é necessário ter instalado em seu computador os programas:

- Compilador C++
- GNU/Octave e seu respectivo pacote de estatísticas
- Gnuplot
- Recomenda-se também o computador com sistema operacional GNU/Linux não sendo garantida a execução dos programas em outro sistema operacional embora o mesmo seja totalmente possível.

A aleatoriedade

Aleatoriedade é uma parte importante dos algoritmos implementados, então afim de melhorar a geração de números aleatórios foram usados comandos especiais que, em tese, geram números aleatórios baseados no hardware do computador.

No entanto, tanto esses comandos especiais quanto os comandos comuns de geração de números aleatórios falharam, não em gerar números aleatórios, mas sim em gerar números aleatórios **iniciais** diferentes dos gerados anteriormente em um curto espaço de

tempo (12 horas). Sendo assim, apesar de quase todos os experimentos convergirem, eles, rigorosamente, convergem sempre em uma quantidade fixa de gerações ou interações em certos horários do dia.

Há que se pesquisar mais profundamente este tópico para que se encontre uma solução para tal problema.

Hill Climbing e Simulated Annealing

Para realização dos experimentos e resolução dos problemas apresentados foram desenvolvidas bibliotecas na linguagem C++. Apenas essas bibliotecas não são suficientes para resolver os problemas, para que os problemas sejam definidos e resolvidos é necessário a criação das **funções de fitness**. É importante frisar que as funções de fitness devem retornar resultados menores para as melhores soluções e resultados maiores para piores soluções.

Algoritmos genéticos

Para realização dos experimentos e resolução dos problemas apresentados foi desenvolvida uma biblioteca na linguagem C++. Esta biblioteca pode ser usada para resolução de qualquer problema que inclua algoritmo genéticos já que a mesma usa como padrão cadeias de números binários para representar o problema a ser resolvido.

É importante notar que esta biblioteca é composta apenas de uma classe e que esta classe em si não é suficiente para definir e resolver o problema. A fim de definir e resolver o problema é necessário criar três funções externas a classe:

print () - Essa função tem como objetivo Mostrar o conteúdo do que está sendo processado,novamente, perceba que essa é uma função que o propositor do problema deve codificar para cada problema específico.

genesisFunction() - Como o próprio nome da função sugere ela é responsável pela criação da população inicial, essa função deve ser implementada de forma que a

população inicial para o problema em questão seja a melhor possível.

fitness() - Essa é a função mais importante de todo o programa. Esta função tem o papel de avaliar cada um dos indivíduos de uma população, novamente, ela deve ser implementada de acordo com o problema proposto. É importante frisar que essa função deve retornar valores menores para resultados melhores, ou seja, quanto melhor o desempenho do indivíduo mais próximo de zero é o valor que a função deve retornar.

Os experimentos

A quantidade máxima de interações em todos os experimentos é 10000, Perceba que esse valor é o máximo, alguns experimentos conseguiram convergir para valores mínimos ou máximos antes desse total de interações.

No caso dos algoritmos genéticos a quantidade máxima de iterações é interpretada como a quantidade máxima de gerações permitidas.

No caso do algoritmo de simulated annealing a quantidade máxima de interações é interpretada como a temperatura inicial do sistema.

Os gráficos foram construídos segundo uma escala logarítmica pois devido a grande quantidade de dados e aos potenciais grandes valores envolvidos, gráficos em escala usual ficariam ilegíveis.

É importante lembrar que quanto menor valor de fitness mais o indivíduo é qualificado para resolver o problema.

Ao lado de cada título ou subtítulo de cada experimento haverão palavras contidas entre parênteses, essas palavras são indicadores de quais classes dentro do código-fonte contém a implementação do experimento.

Experimento 1 (geneticExperiments::Experiment01)

Experimento 1 teve como fim a resolução do problema descrito abaixo:

1) Implemente um Algoritmo Genético para o exemplo de reconhecimento de padrões

apresentado em aula. Em vez de reconhecer o número 1 seu algoritmo deve reconhecer o número 0, representado pela bitstring [1 1 1 1 0 1 1 0 1 1 1 1].

Teste diferentes taxas de crossover e mutação e compare os resultados. Faça experimentos apenas com crossover e apenas com mutação e compare também os resultados.

A quantidade de dados gerada por esse experimento foi extremamente pequena dado a simplicidade do problema e os poucos locus presentes nos “cromossomos” de cada indivíduo.

Tentativa 1:

Parâmetro	**Valor**
População	04
Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,1
Taxa de crossover	0,4
Iterações máximas	10000

Tabela 1: Experimento 1 - Parâmetros da tentativa 1

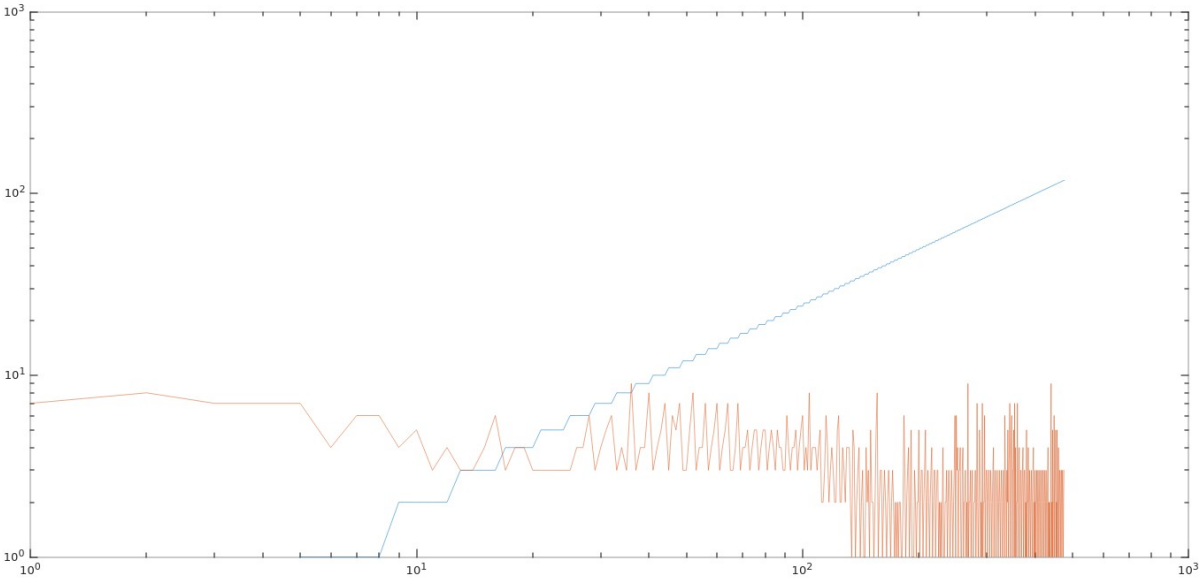

Figura 2: Experimento 1 - Gráfico da tentativa 1

Na figura acima a linha vermelha representa o valor de fitness a linha azul representa a quantidade de gerações.

Nota-se que a população não muda muito até aproximadamente a nona geração, momento no qual a mesma começa a se diversificar. A diversificação é representada pela variação vertical da linha vermelha indicando que existem indivíduos com maior e menor fitness. O pico da diversificação da população se dá aproximadamente após a centésima geração. Percebemos que neste momento os resultados se aproximam dos mínimos desejados sem no entanto prejudicar a existência dos menos aptos, dessa forma garantimos que o algoritmo não explore apenas os máximos locais aumentando assim a probabilidade de encontrar os máximos globais.

O algoritmo resolveu o problema em 118 gerações.

Resultado	**Valor**
x	o padrão procurado
Fenótipo (valor da função de aptidão)	0
Genótipo	111101101111
Primeira convergência	118 gerações
Valor médio da função de aptidão	2,8782
Desvio padrão da função de aptidão	1,6877

Tabela 2: Experimento 1 - Resultados da tentativa 1

Tentativa 2:

Parâmetro	**Valor**
População	04
Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,3
Taxa de crossover	0,4
Iterações máximas	10000

Tabela 3: Experimento 1 - Parâmetros da tentativa 2

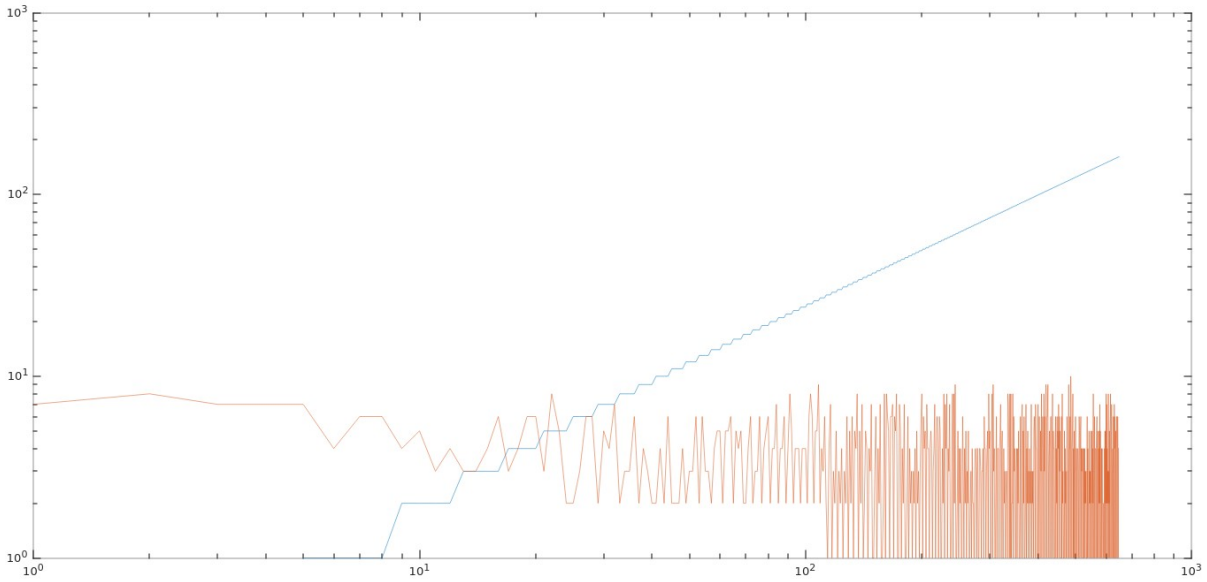

Figura 3: Experimento 1 - Gráfico da tentativa 2

Na figura acima a linha vermelha representa o valor de fitness a linha azul representa a quantidade de gerações.

O comportamento do algoritmo para esta nova taxa de mutação é muito similar ao caso anterior a única mudança significativa foi a quantidade de gerações necessárias para se chegar ao resultado desejado.

O algoritmo resolveu o problema em 161 gerações.

Resultado	Valor
x	o padrão procurado
Fenótipo (valor da função de aptidão)	0
Genótipo	111101101111
Primeira convergência	161 gerações
Valor médio da função de aptidão	3,5123
Desvio padrão da função de aptidão	2,1662

Tabela 4: Experimento 1 - Resultados da tentativa 2

Tentativa 3:

Parâmetro	Valor
População	04
Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,1
Taxa de crossover	0,6
Iterações máximas	10000

Tabela 5: Experimento 1 - Parâmetros da tentativa 3

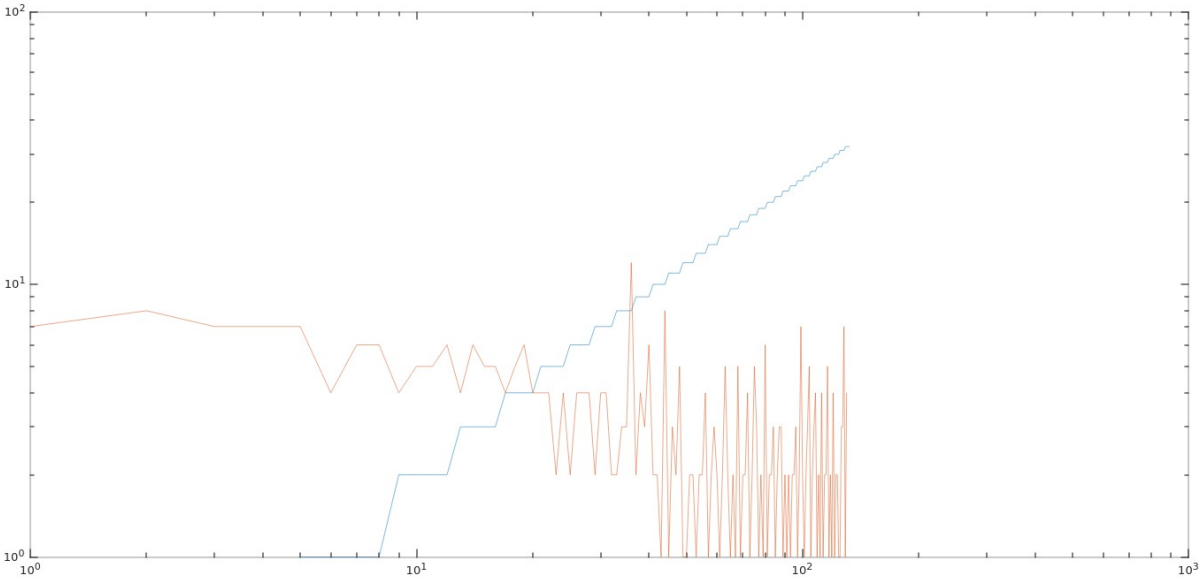

Figura 4: Experimento 1 - Gráfico da tentativa 3

Na figura acima a linha vermelha representa o valor de fitness a linha azul representa a quantidade de gerações.

O comportamento do algoritmo para esta nova taxa de crossover foi surpreendentemente melhor. A diversificação da população se iniciou antes da décima geração e a solução foi encontrada mas rapidamente.

O algoritmo resolveu o problema em 32 gerações.

Resultado	**Valor**
x	o padrão procurado
Fenótipo (valor da função de aptidão)	0
Genótipo	111101101111
Primeira convergência	32 gerações
Valor médio da função de aptidão	3,0379
Desvio padrão da função de aptidão	1,9977

Tabela 6: Experimento 1 - Resultados da tentativa 3

Tentativa 4:

Parâmetro	**Valor**
População	04

Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,3
Taxa de crossover	0,6
Iterações máximas	10000

Tabela 7: Experimento 1 - Parâmetros da tentativa 4

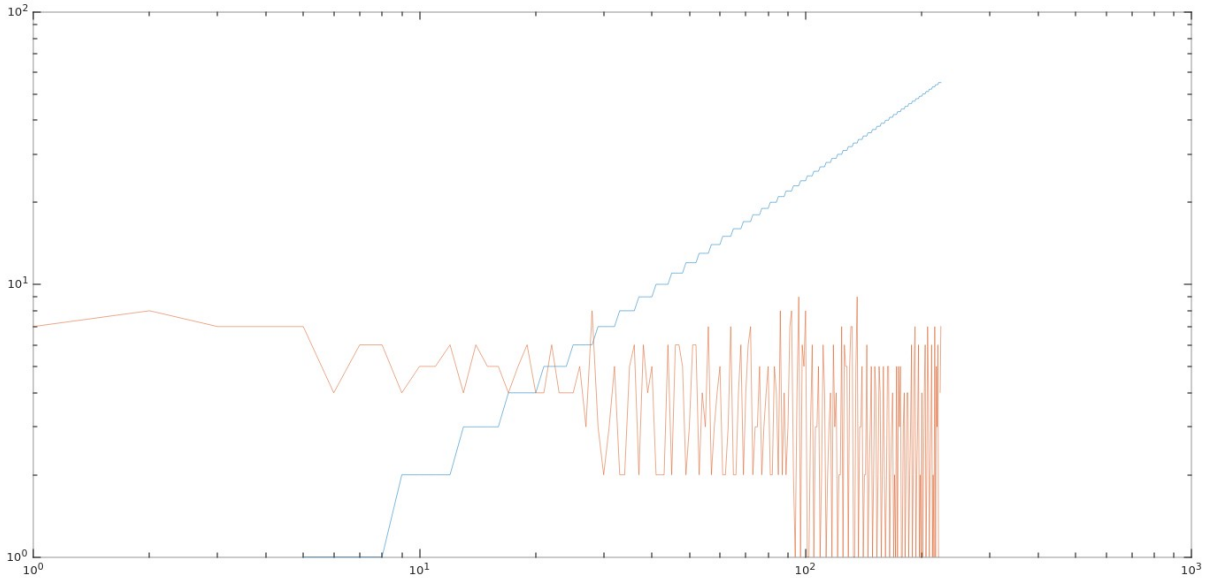

Figura 5: Experimento 1 - Gráfico da tentativa 4

Na figura acima a linha vermelha representa o valor de fitness a linha azul representa a quantidade de gerações.

O comportamento do algoritmo para estas novas taxas de crossover e mutação foi pior que o caso anterior, inicialmente a hipótese era que a mutação seria mais benéfica para convergência do que o crossover, no entanto, essa hipótese se provou falsa. É importante frisar que sem a mutação não haveria a diversificação inicial dos indivíduos, então, apesar da mesma não impactar tanto na convergência ela continua sendo importante. A diversificação da população se iniciou antes da décima geração e a solução foi encontrada mais lentamente.

O algoritmo resolveu o problema em 55 gerações.

Resultado	Valor
x	o padrão procurado
Fenótipo (valor da função de aptidão)	0
Genótipo	111101101111
Primeira convergência	55 gerações
Valor médio da função de aptidão	3,6339
Desvio padrão da função de aptidão	2,0662

Tabela 8: Experimento 1 - Resultados da tentativa 4

Tentativa 5:

Parâmetro	Valor
População	04
Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,1
Taxa de crossover	0,9
Iterações máximas	10000

Tabela 9: Experimento 1 - Parâmetros da tentativa 5

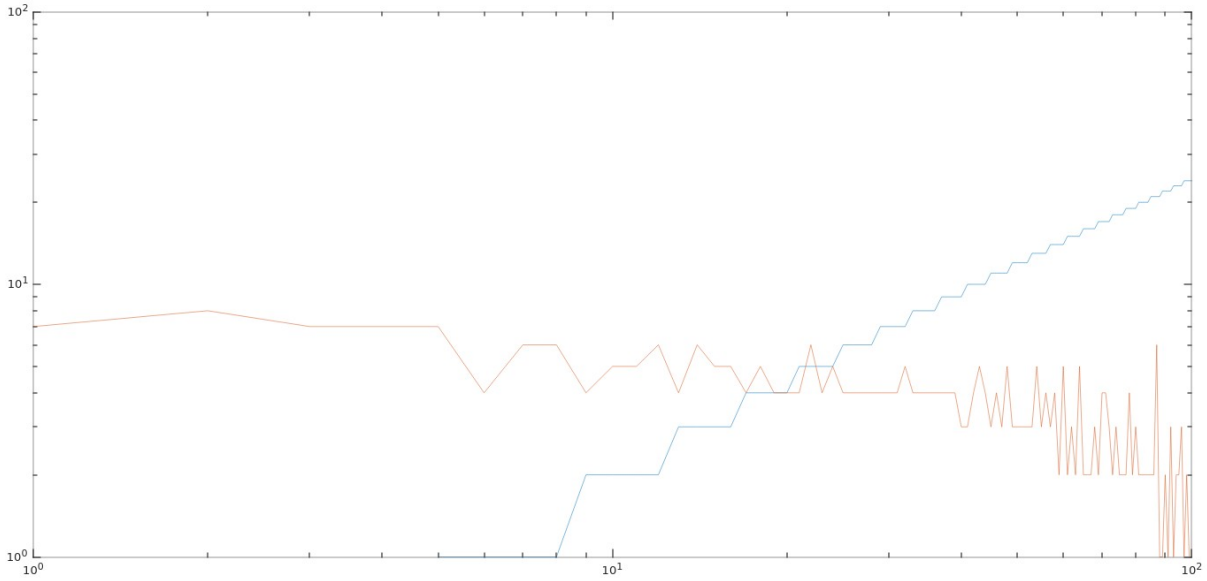

Figura 6: Experimento 1 - Gráfico da tentativa 5

Na figura acima a linha vermelha representa o valor de fitness a linha azul representa a quantidade de gerações.

O comportamento do algoritmo para esta nova taxa de crossover, como o esperado, convergiu rapidamente!

O algoritmo resolveu o problema em 24 gerações.

Resultado	**Valor**
x	o padrão procurado
Fenótipo (valor da função de aptidão)	0
Genótipo	111101101111
Primeira convergência	24 gerações
Valor médio da função de aptidão	3,5600
Desvio padrão da função de aptidão	1,5720

Tabela 10: Experimento 1 - Resultados da tentativa 5

Experimento 2 (Comparativo entre algoritmos genéticos, Hill Climbing e Simulated annealing)

Experimento 2 teve como fim a resolução do problema descrito abaixo:

2)Implemente um Algoritmo Genético para maximizar a função ***g(x)=(2^(-2* ((x−0,1)/0,9)^***

2))* (sin(5πx))^6, *já utilizada nos exercícios feitos em aula. Utilize uma representação de bitstring. Compare o resultado obtido com os resultados que você obteve com os algoritmos Subida da Colina e Recozimento Simulado aplicados a esta mesma função nos exercícios feitos em sala de aula.*

A partir deste ponto um novo desafio se apresentou: Como representar números de ponto flutuante em notação binária. Decidiu-se por usar a representação mais comum para computadores, ou seja, o padrão IEEE 754, e, apesar do tempo considerável gasto na pesquisa e na implementação desse formato, o mesmo não será explicado neste trabalho, para maiores informações consulte as fontes ao final deste documento.

Executando o algoritmo genético (geneticExperiments::Experiment02)

Parâmetro	**Valor**
População	04
Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,1
Taxa de crossover	0,4
Iterações máximas	10000

Tabela 11: Experimento 2 - Parâmetros do algoritmo genético

Os gráficos abaixo mostram a variação dos resultados dada a variação do crossover:

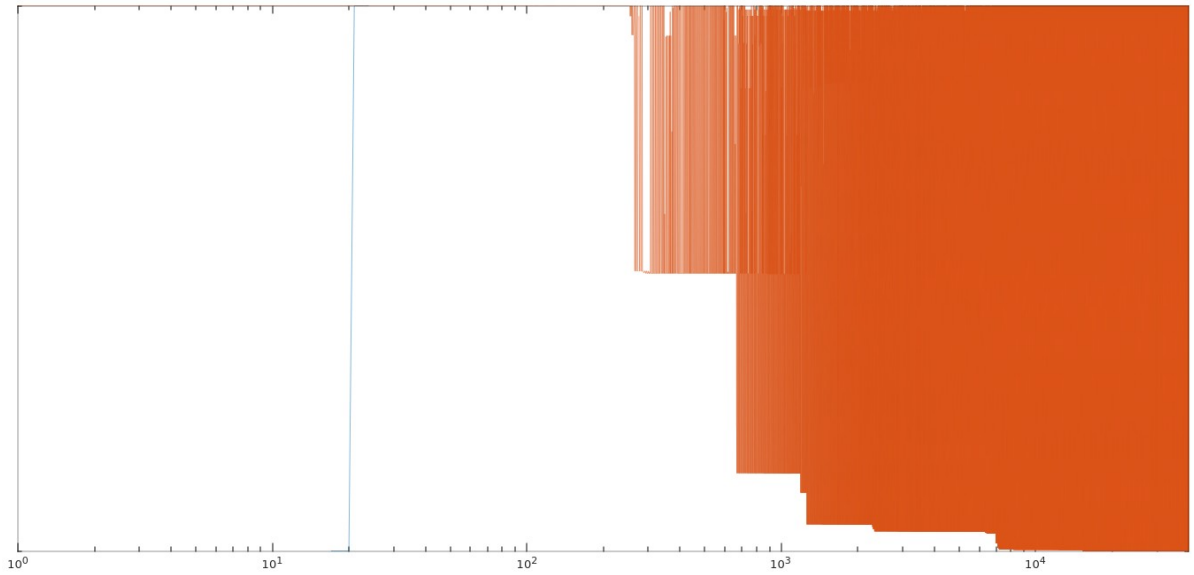

Figura 7: Experimento 2 - Gráfico do algoritmo genético com crossover = 0,4

Taxa de crossover = **0,4**, Valor encontrado: **0,1**, Aplicando-se a função a esse valor se tem: g(0,1)=(2^(-2* ((0,1-0,1)/0,9)^ 2))* (sin(5π0,1))^6 = 1

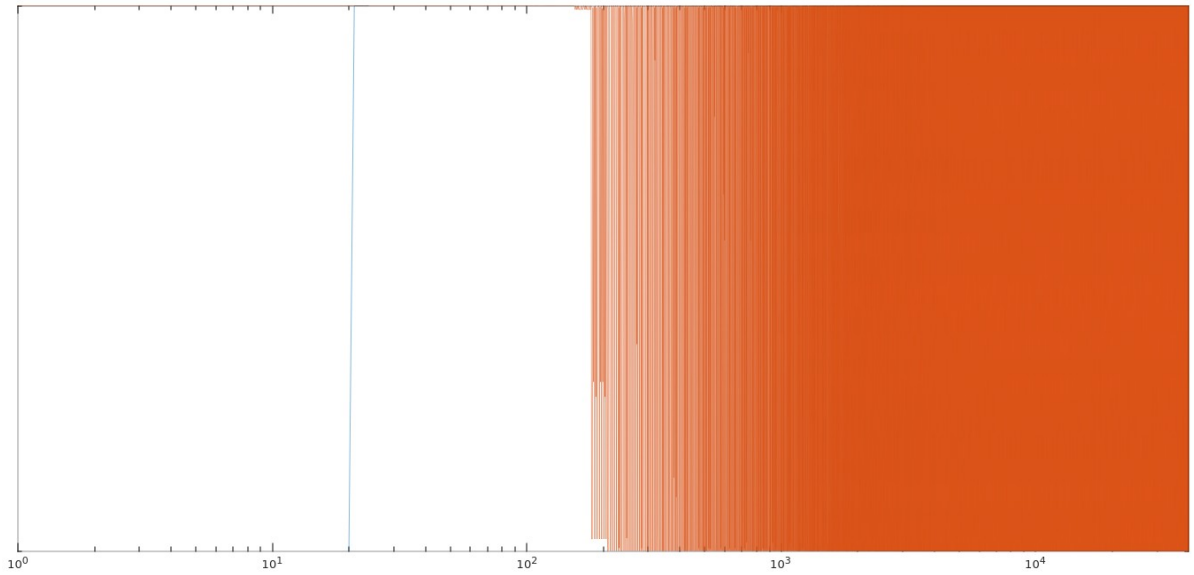

Figura 8: Experimento 2 - Gráfico do algoritmo genético com crossover = 0,9

Taxa de crossover = **0,9**. Valor encontrado: **0,0996094**, Aplicando-se a função é esse valor se tem: g(0,0996094)=(2^(-2* ((0,0996094−0,1)/0,9)^ 2))* (sin(5π0,0996094))^6 = 0,99989.

As figuras acima seguem o mesmo padrão de cores anteriormente citado, no entanto, nesse caso se escolheu limitar o intervalo de representação vertical do gráfico pois a

única dimensão que cresce significativamente na vertical é a dimensão do número de gerações. Como neste trabalho o que importa é avaliar o fitness de cada indivíduo o gráfico foi propositalmente limitado aos menores e maiores valores de fitness.

Dito isso nota-se que uma taxa de crossover maior implica em uma convergência mais rápida, no entanto, isso não garante uma solução mais precisa e nem que a convergência em questão esteja correta.

Quando a taxa de crossover é de 0,9 apesar do algoritmo realizar todas as 10000 gerações o valor dos indivíduos não consegue convergir para o máximo desejado, por outro lado, quando a taxa de crossover é 0,4 o algoritmo converge para a solução ótima em 4972 gerações.

Resultado	**Valor**
x	1,000000
Fenótipo (valor da função de aptidão)	1,000007
Genótipo	00111101110011010000000000 000000
Primeira convergência	4972 gerações
Valor médio da função de aptidão	0,090989
Desvio padrão da função de aptidão	0,14758

Tabela 12: Experimento 2 - Resultados do algoritmo genético

Executando os algoritmos Hill Climbing

Hill Climbing estocástico (hillClimbingAnnealingExperiments::Experiment02)

O primeiro algoritmo não genético usado para resolver ***g(x)=(2^(-2* ((x−0,1)/0,9)^ 2))* (sin(5πx))^6*** será o algoritmo de Hill Climbing estocástico. De forma alguma o Hill Climbing estocástico parece convergir, mesmo após várias verificações no algoritmo a procura de erros, testes com diferentes funções de fitness e execuções em modo debug esse algoritmo parece incapaz de convergir para algum valor.

Dados do experimento

Parâmetro	**Valor**
Máximo de iterações	10000
Perturbação	[0 .. 1 [

Tabela 13: Experimento 2 - Parâmetros do Hill Climbing estocástico

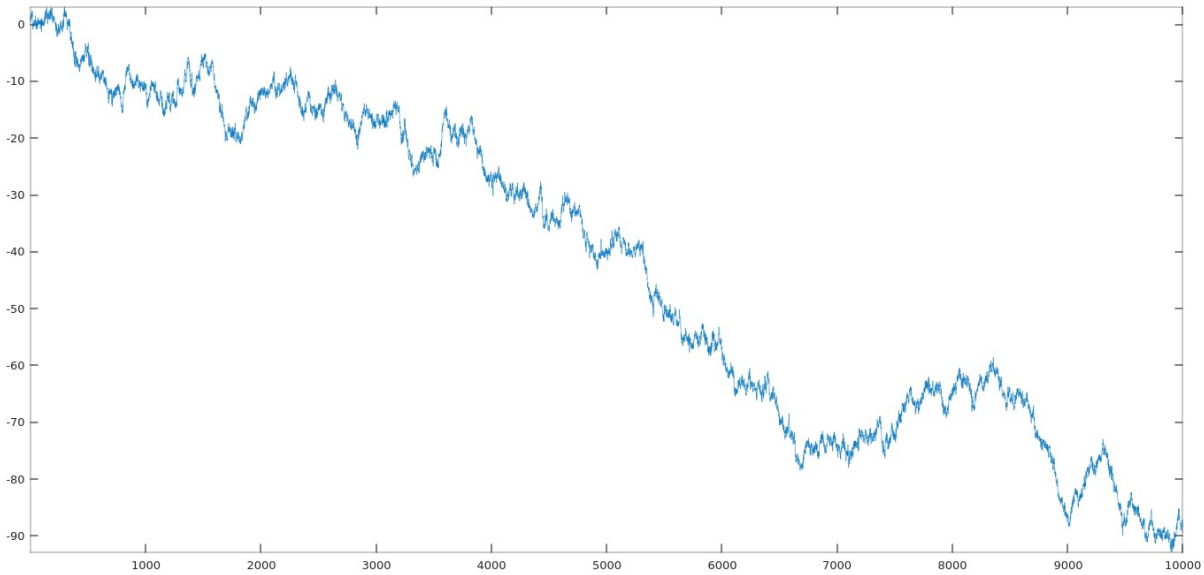

Figura 9: Experimento 2 - Gráfico do Hill Climbing Estocástico

Como se pode perceber no gráfico acima os valores gerados por este algoritmo são basicamente divergentes não havendo de forma alguma convergência. (A escala deste gráfico não é logarítmica). Valor da solução a este algoritmo chegou é -88.9225, valor este muito longe do ideal.

Resultados:

Resultado	**Valor**
x	-88,9225
Primeira convergência	não converge
Valor médio da função de aptidão	32,841
Desvio padrão da função de aptidão	21,817

Tabela 14: Experimento 2 - Resultados Hill Climbing Estocástico

Hill Climbing (hillClimbingAnnealingExperiments::Experiment05)

Dados do experimento

Parâmetro	**Valor**
Máximo de iterações	10000
Perturbação	[0 .. 1 [

Tabela 15: Experimento 2 - Parâmetros do Hill Climbing

Ao contrário do Hill Climbing estocástico este algoritmo convergiu de forma muito eficiente

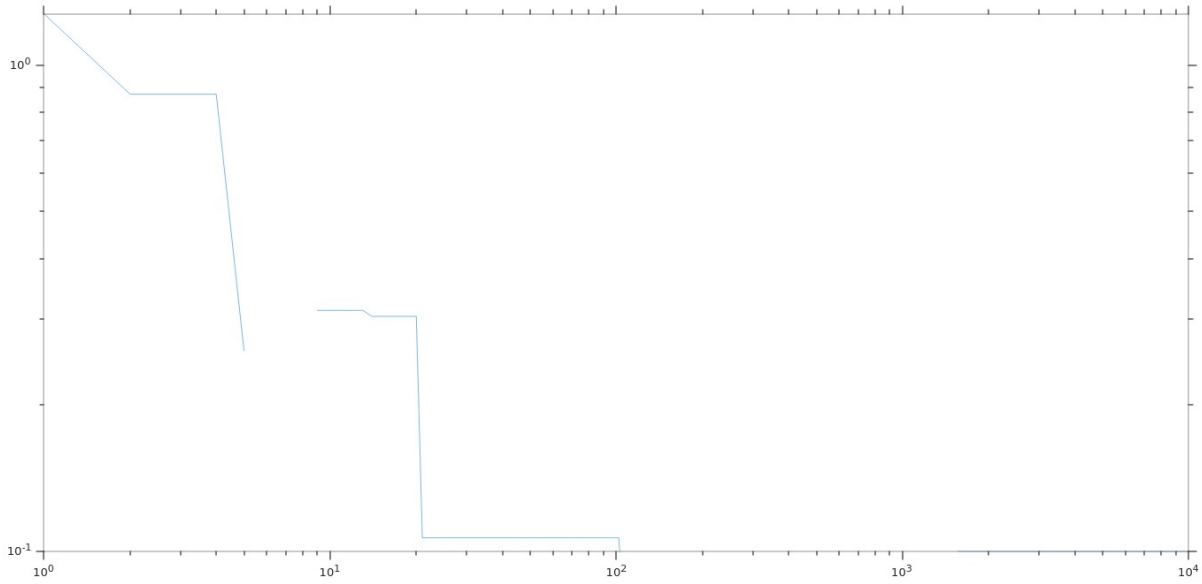

Figura 10: Experimento 2 - Gráfico do Hill Climbing

Olhando o gráfico podemos perceber que a convergência do Hill Climbing foi extremamente rápida. O algoritmo chegou ao valor **0,100016** em apenas 1988 iterações. Dependendo do fator de perturbação Hill Climbing pode convergir mais rápido, perdendo um pouco de sua precisão ou mais lentamente tendo como contrapartida uma precisão maior. De qualquer forma provou ser um algoritmo extremamente eficiente quando o espaço de procura não é tão complexo.

Resultados:

Resultado	**Valor**
x	0,100016
Primeira convergência	1988
Valor médio da função de aptidão	0,090989
Desvio padrão da função de aptidão	0,14758

Tabela 16: Experimento 2 - Resultados Hill Climbing

Simulated Annealing (hillClimbingAnnealingExperiments::Experiment04)

O mais exigente em termos de processamento de todos os algoritmos. Neste algoritmo o número máximo de iterações é interpretado como a temperatura inicial do sistema, temperatura essa que vai diminuindo gradualmente a cada iteração.

Dados do experimento

Parâmetro	**Valor**
Temperatura inicial do sistema	10000
Taxa de resfriamento	0,99999
Temperatura de parada	0,0001

Tabela 17: Experimento 2 - Parâmetros do Simulated Annealing

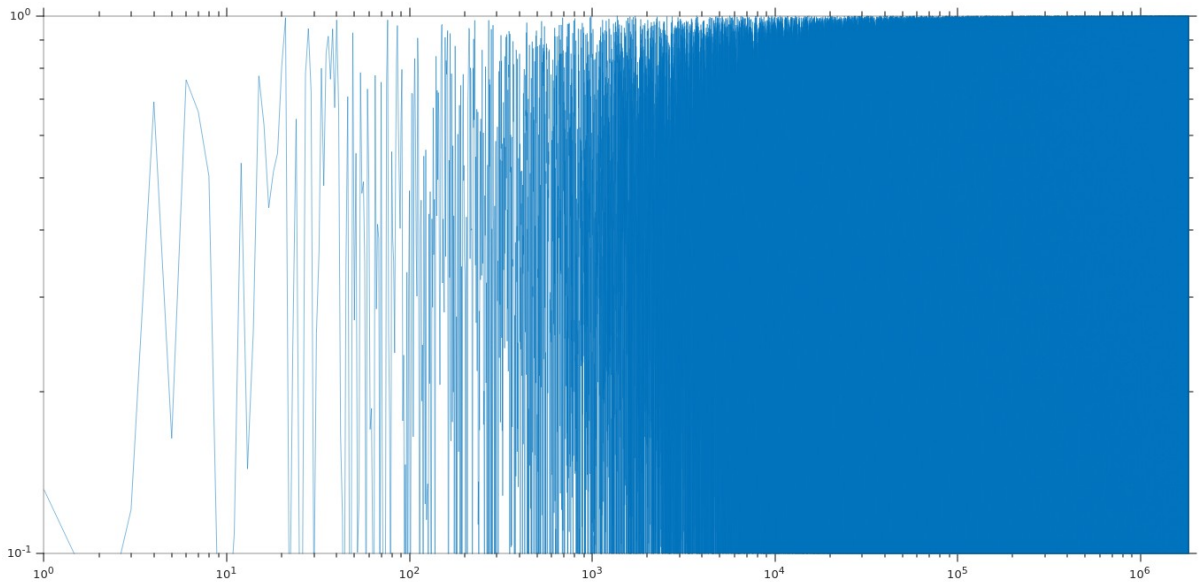

Figura 11: Experimento 2 - Gráfico do Simulated Annealing

Resultados:

Resultado	**Valor**
x	0,115999
Primeira convergência	81931
Valor médio da função de aptidão	0,090989
Desvio padrão da função de aptidão	0,14758

Tabela 18: Experimento 2 – Resultados Simulated Annealing

Perceba que o valor médio da função de aptidão e o valor do desvio padrão são os mesmos do experimento anterior. Isso não é uma coincidência! Dada uma série muito longa de valores similares avaliados por uma mesma função de aptidão é esperado que as estatísticas coincidam.

Experimento 3 (geneticExperiments::Experiment04)

Experimento 3 teve como fim a resolução do problema descrito abaixo:

3) Utilize um Algoritmo Genético para minimizar a seguinte função no intervalo contínuo

[-5, +5]

[-5, +5]

f (x, y) = (1 − x)^2 + 100(y − x^2)^2

Como já foi citado anteriormente problemas que envolvem números de ponto flutuante foram resolvidos usando-se a representação binária IEEE 754, no entanto, nesse problema um desafio a mais se pôs: Como representar 2 números de ponto flutuante de uma única vez? A solução mais simples e óbvia foi aplicada: Juntou-se as duas cadeias de números binários em uma só resultado em uma cadeia de 64 bits.

Dados do experimento

Parâmetro	**Valor**
População	04
Influência de fitness	08
Influência da diversidade	02
Taxa de mutação	0,1
Taxa de crossover	0,4
Iterações máximas	10000

Tabela 19: Experimento 3 - Parâmetros

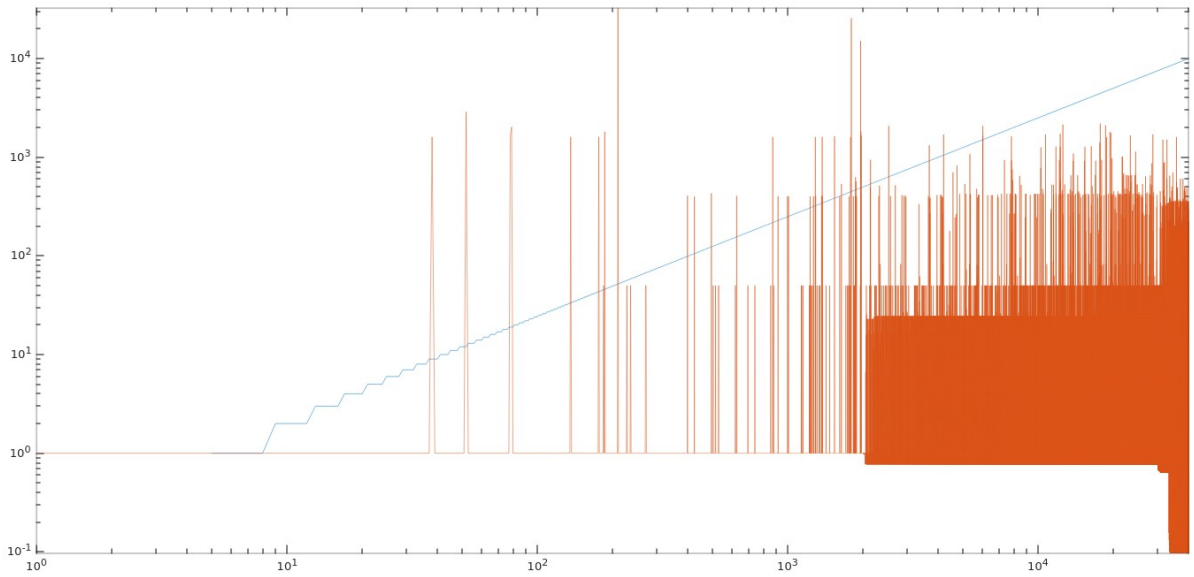

Figura 12: Experimento 3 - Gráfico

Importante: Alguém que observe com mais atenção perceberá que a escala da gerações passa da quantidade de iterações máximas 10000, isso se dá pois cada genótipo na prática gera dois indivíduos dobrando a quantidade de dados de cada geração.

Dada a complexidade do problema a convergência, como se pode perceber no gráfico, Iniciou-se um pouco tardiamente. Nota-se que a diversidade dos indivíduos começou aumentar aproximadamente após a centésima geração.

É interessante notar que por muitas gerações gerou-se apenas diversidade e não

convergência, a mesma surgiu apenas nas últimas gerações e de forma abrupta.

Resultados:

Resultado	**Valor**
x	1,000000
y	1,000366
Fenótipo (valor da função de aptidão)	0,000013
Genótipo	00111111011111111111111111 11111100111111100000000000 110000000000
Primeira convergência	9457 gerações
Valor médio da função de aptidão	0,090989
Desvio padrão da função de aptidão	0,14758

Tabela 20: Experimento 3 - Resultados

Referências

C++ Reference

http://www.cplusplus.com/reference/

Calculando o desvio padrão no Octave

http://www.basef.com.br/old/octave-matlab/514-2014-07-09-20-58-31

2D & 3D Plots

http://www.malinc.se/math/octave/threeden.php

Octave Tutorial #4 - Plotting Data

https://www.youtube.com/watch?v=B29EftnPbWo&list=PL1A2CSdiySGJ6oZe6XB-TTCFuHc5Fs1PO&index=4

Calling .csv file into Octave

https://stackoverflow.com/questions/25325577/calling-csv-file-into-octave

Padrão IEEE 754

https://www.youtube.com/watch?v=PDgT0T0Yodo&t=476s

Decimal to IEEE 754 Floating Point Representation

https://www.youtube.com/watch?v=8afbTaA-gOQ&t=316s