Algoritmo Genético

Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)

Departamento de Estatística, Matemática Aplicada e Computação (DEMAC)

Curso: Pós-Graduação em Ciência da Computação

Disciplina: Computação Inspirada pela Natureza

Prof.: Fabrício Breve – Trabalho no 1 – Data de Entrega: 14/05/2020

Nome: Lucas de Brito Silva

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.

Os testes foram realizados usando timestamp como semente, para que depois fosse possível reproduzir novamente os mesmos, visto que o algoritmo é estocástico e possui vários parâmetros randômicos. Vale ressaltar que dentre esses parâmetros, temos:

- Hipótese de *crossover*;
- Ponto de *crossover*;
- Hipótese de mutação;
- Posição das agulhas para a roleta estocástica.

Além disso, vale ressaltar que por padrão, as populações contaram com 8 indivíduos aleatórios entre 0 e 4095, que são representados por uma *bitstring* de 12 bits, além do '0b' que é o padrão para a representação de binários em Python.

Para poder calcular a aptidão dos indivíduos em cada geração foi utilizado da diferença entre o tamanho de cada individuo e a distância de Hamming.

A seleção foi feita com a técnica de amostragem universal estocástica (SUS) de modo que a roleta foi composta por 8 agulhas que foram distribuídas igualmente (com a mesma distância) entre a roleta de 360º, além de ser "girada" apenas uma vez.

Na reprodução o ponto de crossover foi de 0,6, em que para que houvesse crossover, era necessária uma hipótese aleatória menor que o valor supracitado.

Para o ponto de mutação foi definido um valor de 0,02, em que cada bit que compunha uma população era submetido a uma hipótese de passar por mutação e a mesma ocorria quando a hipótese era menor ou igual ao valor supracitado.

Testes de algoritmo genético completo

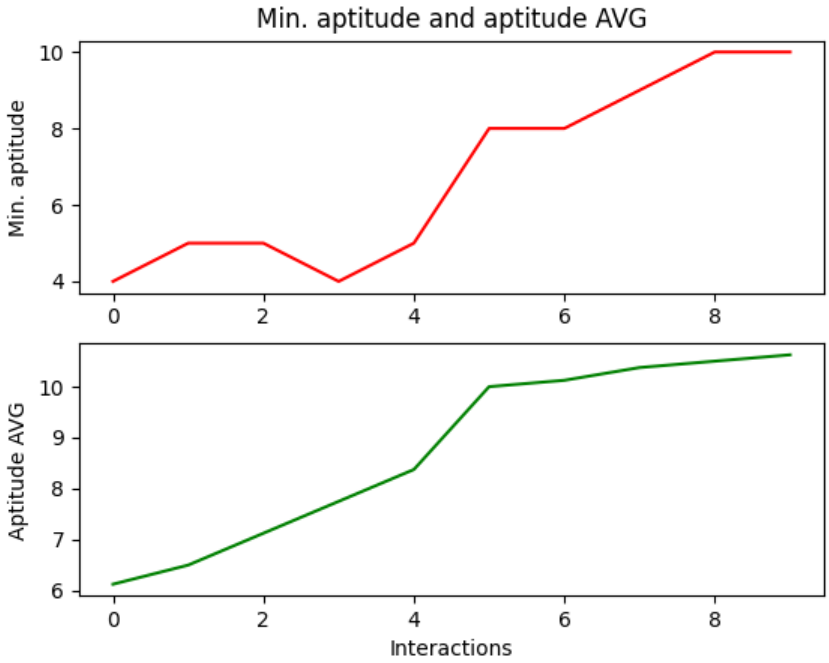
Os testes a seguir contaram tanto com o crossover, quanto com a mutação.

Teste 1

Interactions: 11	Seed: 1589160948.3090756	Time: 0.002920389175415039

Population: ['0b100101100100', '0b111010011110', '0b110010000110', '0b010010000101', '0b011100101000', '0b100111010101', '0b100011101110', '0b101100011111']

Final Generation: ['0b111100001111', '0b111100101111', '0b111101101111', '0b111100101010', '0b111100101111', '0b111100101111', '0b111100101101', '0b111100101111']



Teste 2

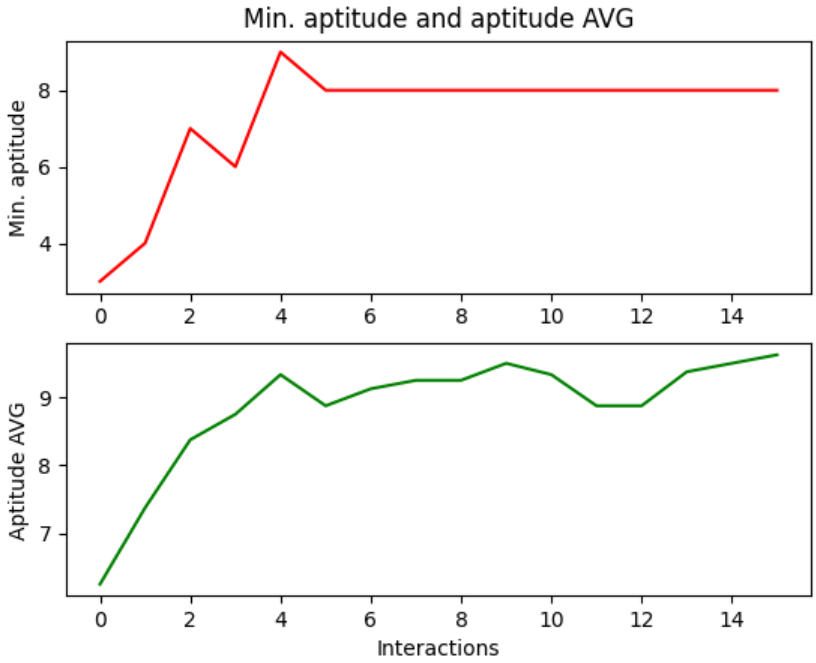
Interactions: 17

Seed: 1589161210.455861

Time: 0.0025098323822021484

Population: ['0b000010111110', '0b011001100101', '0b100011101111', '0b100010010101', '0b000001001010', '0b000100100100', '0b101001001100', '0b111101000111']

Final Generation: ['0b111101101111', '0b111000101100', '0b111001101110', '0b111001101110', '0b111001101101', '0b111001101110']



Teste 3

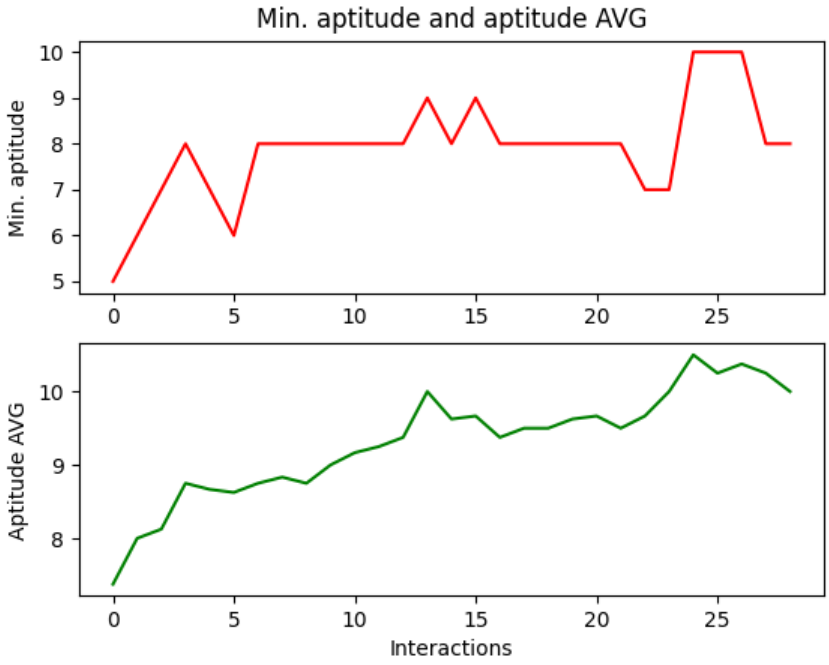
Interactions: 30

Seed: 1589161704.3551645

Time: 0.0040798187255859375

Population: ['0b000001101100', '0b101100001110', '0b011111101110', '0b001100010101', '0b111101001011', '0b100111000111', '0b100111001001', '0b010101001101']

Final Generation: ['0b111100101111', '0b111101101111', '0b011101101111', '0b011101111111', '0b111111101110', '0b101100001110', '0b111100101111', '0b111100101111']



Teste 4

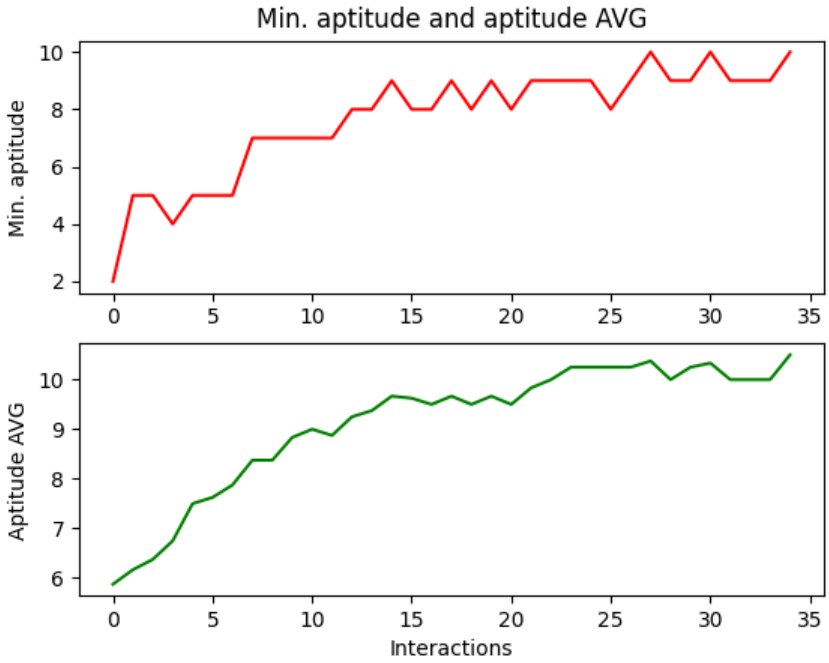
Interactions: 36

Seed: 1589162032.893142

Time: 0.008337020874023438

Population: ['0b101111100010', '0b010010000111', '0b101000101011', '0b010110001100', '0b101011000110', '0b001010010010', '0b100001111101', '0b101100111001']

Final Generation: ['0b111111101011', '0b111101001011', '0b111101101111', '0b011101101110', '0b111101101010', '0b111101101010', '0b111101101011', '0b111101101011']



Teste 5

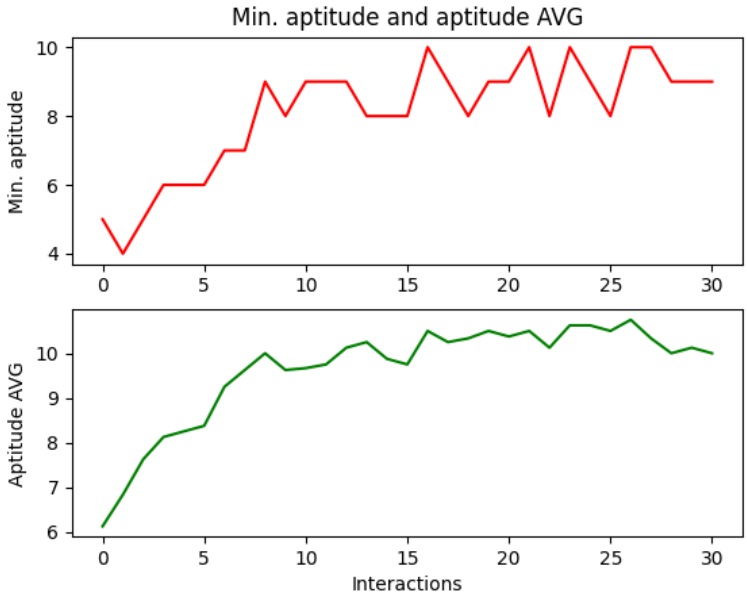
Interactions: 32

Seed: 1589162132.5001028

Time: 0.004332065582275391

Population: ['0b110110011110', '0b010100010011', '0b001010101001', '0b100100100001', '0b001001100100', '0b001001111000', '0b110101100111', '0b100011100011']

Final Generation: ['0b110001101110', '0b110101101111', '0b111000101111', '0b111000101111', '0b111011101111', '0b101001101111', '0b111101101111', '0b111001001111']



Testes de algoritmo genético com crossover

Nessa etapa os testes contaram apenas com crossover. Todavia foi observado, sem exceções que todos eles passavam de 3 minutos de execução e não chegavam ao objetivo, dessa forma para poder ter uma noção de evolução do algoritmo, foi programado uma condição de parada, em que após 100.000 evoluções o algoritmo desse um resultado.

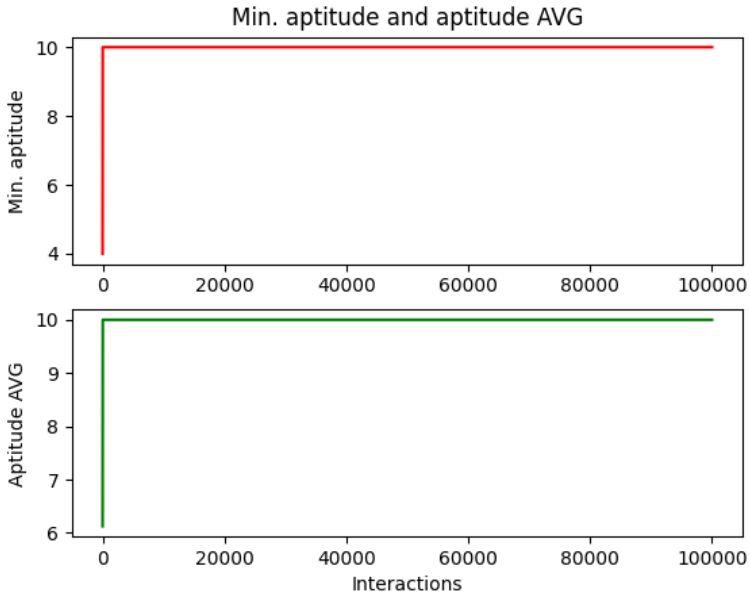
Com excessão do teste 3, que teve um resultado melhor do que quando aplicado mutação, foi possivel observar que todos eles chegavam em um ótimo logal e não podiam evoluir mais para um ótimo global que era o valor desejado.

Teste 1

Interactions: 100000	Seed: 1589160948.3090756	Time: 21.92744207382202

Population: ['0b100101100100', '0b111010011110', '0b110010000110', '0b010010000101', '0b011100101000', '0b100111010101', '0b100011101110', '0b101100011111']

Final Generation: ['0b111001101110', '0b111001101110', '0b111001101110', '0b111001101110', '0b111001101110', '0b111001101110', '0b111001101110', '0b111001101110']

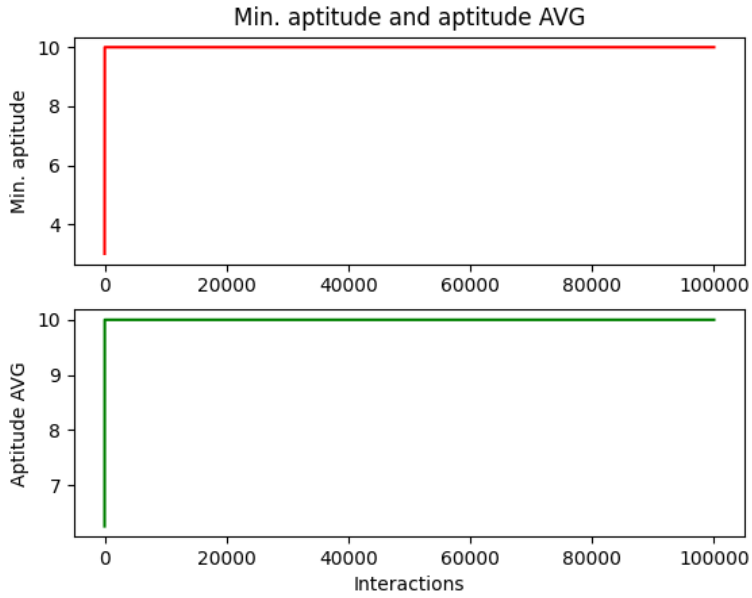


Teste 2

Interactions: 100000 Seed: 1589161210.455861 Time: 27.001359701156616

Population: ['0b000010111110', '0b011001100101', '0b100011101111', '0b100010010101', '0b000001001010', '0b000100100100', '0b101001001100', '0b111101000111']

Final Generation: ['0b111101000111', '0b111101000111', '0b111101000111', '0b111101000111', '0b111101000111', '0b111101000111', '0b111101000111', '0b111101000111']

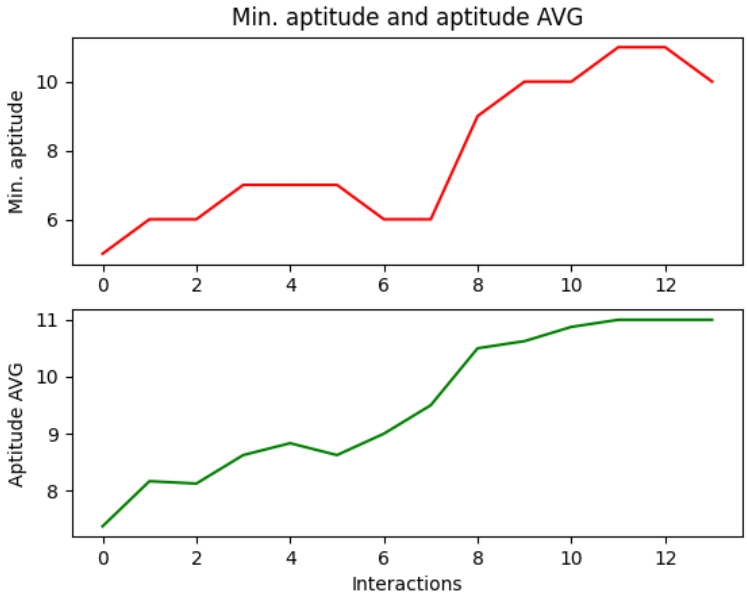


Teste 3

Interactions: 14 Seed: 1589161704.3551645 Time: 0.0047571659088134766

Population: ['0b000001101100', '0b101100001110', '0b011111101110', '0b001100010101', '0b111101001011', '0b100111000111', '0b100111001001', '0b010101001101']

Final Generation: ['0b110101101111', '0b111101001111', '0b110101101111', '0b110101101111', '0b111101101111', '0b110101101011', '0b110101101111', '0b110101101111']

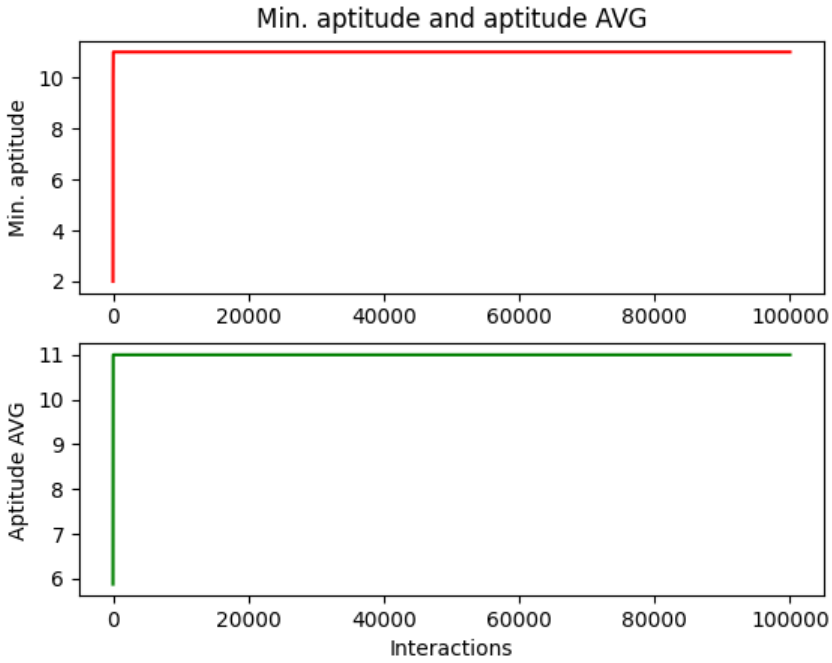


Teste 4

Interactions: 100000 Seed: 1589162032.893142 Time: 22.32247567176819

Population: ['0b101111100010', '0b010010000111', '0b101000101011', '0b010110001100', '0b101011000110', '0b001010010010', '0b100001111101', '0b101100111001']

Final Generation: ['0b101101101111', '0b101101101111', '0b101101101111', '0b101101101111', '0b101101101111', '0b101101101111', '0b101101101111', '0b101101101111']

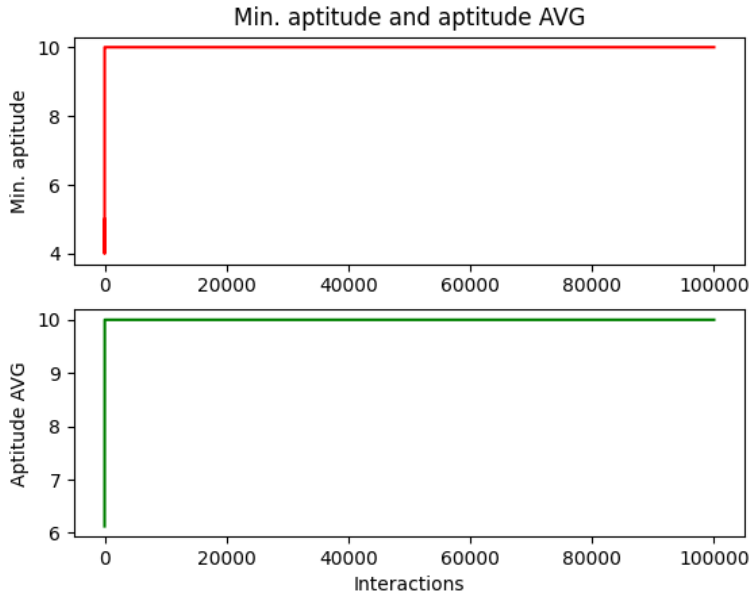


Teste 5

Interactions: 100000 Seed: 1589162132.5001028 Time: 30.218552827835083

Population: ['0b110110011110', '0b010100010011', '0b001010101001', '0b100100100001', '0b001001100100', '0b001001111000', '0b110101100111', '0b100011100011']

Final Generation: ['0b110101100111', '0b110101100111', '0b110101100111', '0b110101100111', '0b110101100111', '0b110101100111', '0b110101100111', '0b110101100111']



Testes de algoritmo genético com mutação

Nessa etapa os testes contaram apenas com o uso da mutação, sem que tivesse uma reprodução precedente. Porém, diferente do esperado, por conta da experiência com o teste utilizando apenas crossover, foi possível desfrutar de resultados de convergência com muito menos evoluções, o que leva a conclusão empírica de que o algoritmo tem ótimos resultados com uma influência maior da mutação do que do crossover.

Teste 1

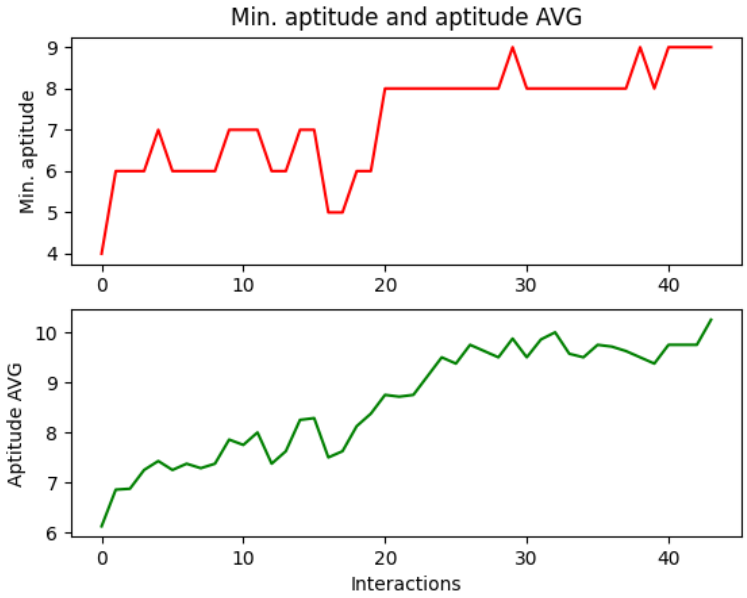
Interactions: 44

Seed: 1589160948.3090756

Time: 0.015102863311767578

Population: ['0b100101100100', '0b111010011110', '0b110010000110', '0b010010000101', '0b011100101000', '0b100111010101', '0b100011101110', '0b101100011111']

Final Generation: ['0b101101111110', '0b001101101110', '0b001101101111', '0b111101101111', '0b111101100110', '0b111001101101', '0b101101101111', '0b101101101111']



Teste 2

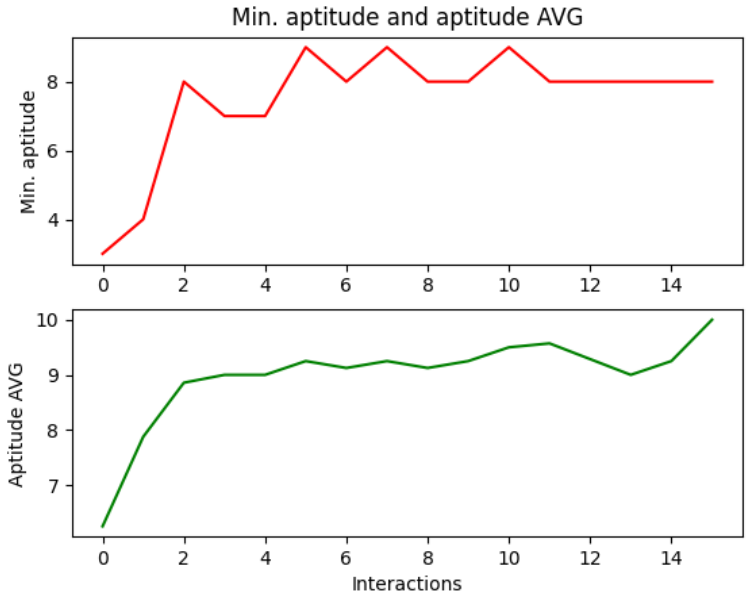
Interactions: 16

Seed: 1589161210.455861

Time: 0.0061604976654052734

Population: ['0b000010111110', '0b011001100101', '0b100011101111', '0b100010010101', '0b000001001010', '0b000100100100', '0b101001001100', '0b111101000111']

Final Generation: ['0b111101100110', '0b011101000101', '0b101101001111', '0b111101001111', '0b111101000110', '0b011101000111', '0b111101001111', '0b111101101111']



Teste 3

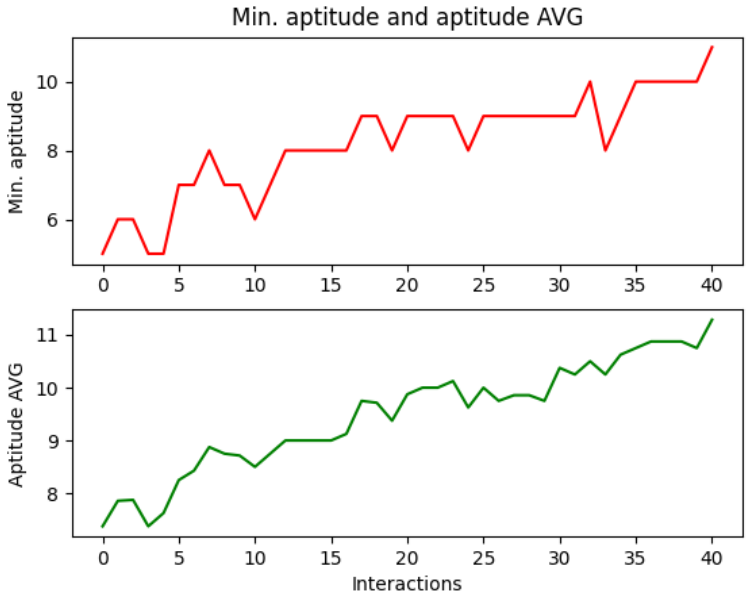
Interactions: 41

Seed: 1589161704.3551645

Time: 0.030269622802734375

Population: ['0b000001101100', '0b101100001110', '0b011111101110', '0b001100010101', '0b111101001011', '0b100111000111', '0b100111001001', '0b010101001101']

Final Generation: ['0b111101101111', '0b111101101011', '0b111101101011', '0b111101101011', '0b111101101111', '0b111101101011', '0b111101101011']



Teste 4

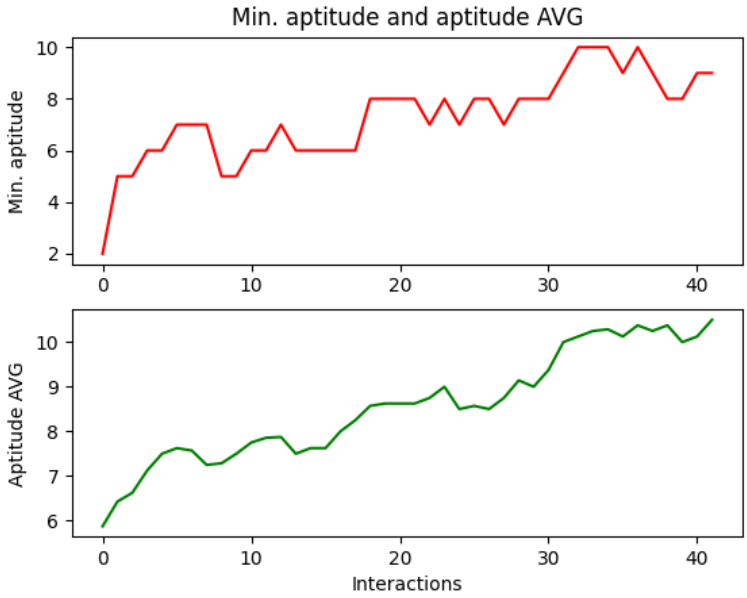
Interactions: 42

Seed: 1589162032.893142

Time: 0.04236912727355957

Population: ['0b101111100010', '0b010010000111', '0b101000101011', '0b010110001100', '0b101011000110', '0b001010010010', '0b100001111101', '0b101100111001']

Final Generation: ['0b111101111110', '0b111101101111', '0b111001111111', '0b101111101111', '0b011101111011', '0b111101111111', '0b101101101111', '0b101101101111']



Teste 5

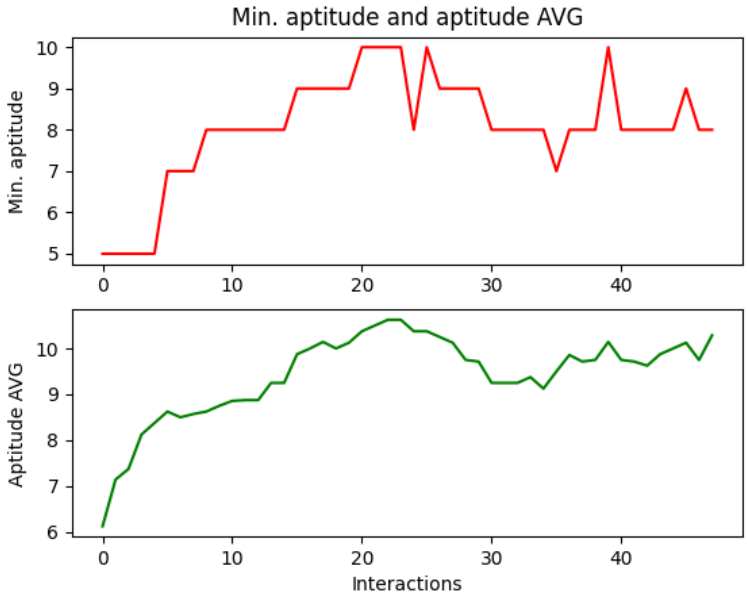
Interactions: 48

Seed: 1589162132.5001028

Time: 0.02854609489440918

Population: ['0b110110011110', '0b010100010011', '0b001010101001', '0b100100100001', '0b001001100100', '0b001001111000', '0b110101100111', '0b100011100011']

Final Generation: ['0b011101101110', '0b111101101111', '0b001100101011', '0b111101101110', '0b011100101111', '0b111100101111', '0b101101111111']



2) Implemente um Algoritmo Genético para maximizar a função g(x), 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. Aproveite para explorar diferentes formas de seleção.

Para a execução desse algoritmo utilizou-se de float em vez de uma bitstring para representar os indivíduos, sendo que a conversão de float para bitstring ocorria quando era necessário utilizar do crossover e logo após a realização do mesmo, a conversão ocorria de bitstring para float novamente.

Assim como no exercício anterior, a probabilidade crossover manteve-se em 0,6 visto que não só em aula, mas também no livro de COLE (2009) essa probabilidade manteve-se, dessa forma coloquei 0,6 como uma probabilidade padrão.

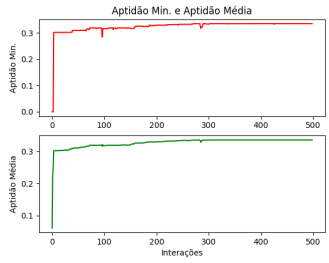
Quanto a mutação, também foi estabelecido um padrão de 0.02, visto que podemos ver questões em que esse valor varia de no minimo 0.015 a 0.02, como no seguinte link:

Why is the mutation rate in genetic algorithms very small?

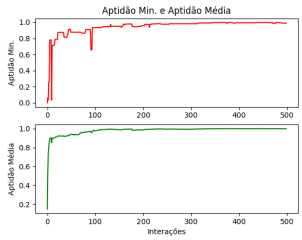
I read somewhere that mutation probability should be nearly 0.015 to 0.02. I wonder how this mutation rate will make any difference to the original chromosome? Here, we present two

https://www.researchgate.net/post/Why_is_the_mutation_rate_in_genetic_algorithms_very_small

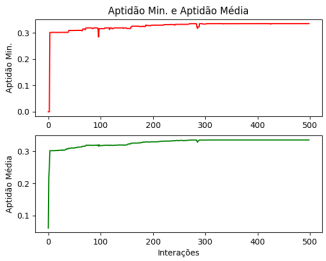
Quanto às populações, foram selecionados 8 indivíduos para cada geração, número o qual foi selecionado empiricamente em um primeiro momento. Porém, demonstrou uma convergência mais efetiva após testes com 8, 20 e 40 indivíduos (utilizando a mesma semente), que são representados nas imagens a seguir, respectivamente.



8 individos



20 individos



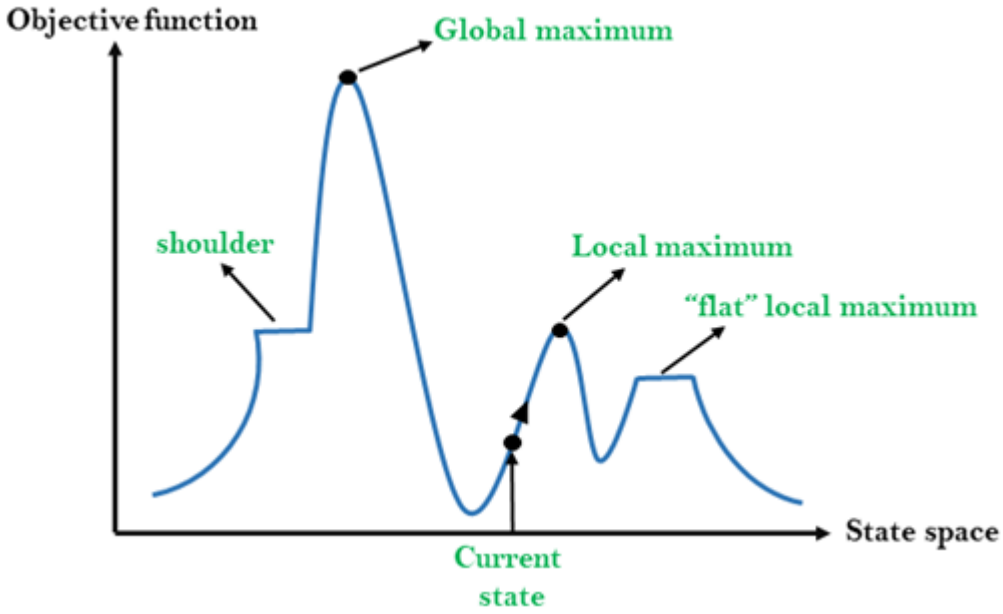
40 individos

Como fator de limitação para o algoritmo genético foi utilizado a quantidade de gerações que o algoritmo teria. No caso, como default deixei um padrão de 500 interações, o que foi mantido também para o hill climbing que foi utilizado de comparativo e será explanado a seguir.

A seleção escolhida para o algoritmo genético foi a amostragem universal estocástica, contando com uma roleta com o número de agulhas igual ao número de indivíduos. A hipótese de escolha para os indivíduos foram determinadas de acordo com a aptidão de cada um, por fim, a roleta foi girada uma única vez, tendo as agulhas apontando para os selecionados para compor uma nova geração.

O Hill Climbing também é um algoritmo estocástico e a partir dessa premissa, sabe-se que o mesmo conta com diversos valores aleatórios e que em cada execução pode obter comportamentos diferentes para alcançar um valor ótimo. Destaca-se no Hill Climbing que o seu objetivo nem sempre encontra um ótimo global, podendo contentar-se com um ótimo local, dessa forma, quando o algoritmo não consegue evoluir, comumente é gera-se uma constante (denominada de flat local maximum,

segundo a figura abaixo) considerando esse valor como melhor local, após uma quantidade x de interações.



Para que o custo computacional fosse reduzido, tanto no algoritmo genético, quanto no hill climbing, foi estabelecido que após ter um valor de custo ou aptidão repetida por um terço do máximo de interações, o algoritmo determinaria aquele valor como ótimo.

Quanto ao Simulated Annealing, essa técnica probabilística foi inspirada no recozimento de metais com a lei física da entropia. Essa técnica tenta, através de interações e uma perturbação dos pontos, encontrar um ótimo global de uma fórmula, sendo que para isso conta com uma redução de sua temperatura. Sua principal diferença do hill climbing em âmbito de desenvolvimento, está na sua função de avaliação, em que a partir de uma hipótese randômica, concede uma probabilidade euleriana de um valor ser selecionado, mesmo quando não apresenta evolução comparada a interação anterior.

Um ponto interessante é que, em tese, com a redução da temperatura a probabilidade fica cada vez mais rigorosa, impedindo de selecionar valores piores que os anteriores o que estabelece, comumente, uma inclinação menor das curvas ou uma elevação grande nos gráfico de representação, todavia ao desenvolver isso não ocorria, pois a função euleriana estabelecia um espécie de constante, dessa forma, foi aplicado uma penalidade à função euleriana que estabelecia a probabilidade de pióra do algoritmo e essa penalidade reduzia de acordo com o resfriamento da solução.

A temperatura usada como padrão no algoritmo Simulated Annealing foi de 500º e foi escolhida com base no máximo de interações definido para os outros dois algoritmos.

Referência citada: COLE, Eric AB. Mathematical and Numerical Modelling of Heterostructure Semiconductor Devices: From Theory to Programming. Springer Science & Business Media, 2009.

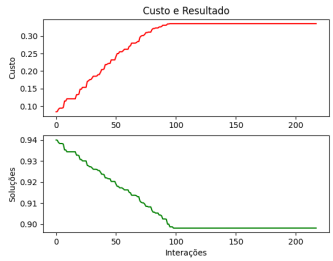
Testes com Hill Cimbing, Simullated Annealing e Algoritmo Genético

Os resultados de solução nas colunas do algoritmo genético nem sempre são os últimos, todavia, são os melhores expressados no algoritmo. Diferente da aptidão média, expressa na coluna "custo", que sempre mostra a aptidão média do último registro.

Teste 1

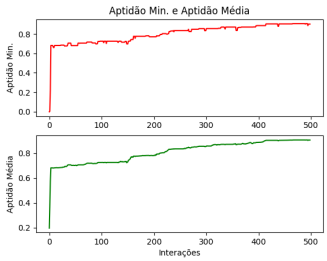
Aa Método	Solução (x)	Custo	Tempo
Hill Climbing	0.8981264940041169	0.33527039134018877	0.002674102783203125
Simulated Annealing	0.8981428432938988	0.3352705575846276	0.020873308181762695
Genetic Algorithm	0.30600405196296165	0.9048457661946905	0.17138457298278809

Semente: 1589467894.853302



Hill Climbing

Simullated Annealing

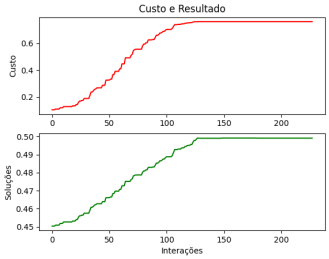


Algoritmo genético

Teste 2

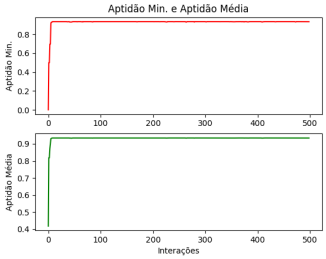
Aa Método	Solução (x)	Custo	Tempo
Hill Climbing	0.49907794607781	0.760937338215296	0.009078025817871094
Simulated Annealing	0.49908916665647757	0.7609372595914304	0.04891085624694824
Genetic Algorithm	0.3005416232637749	0.9337229333376951	0.2226858139038086

Semente: 1589469370.2762575



Hill Climbing

Simulated Annealing

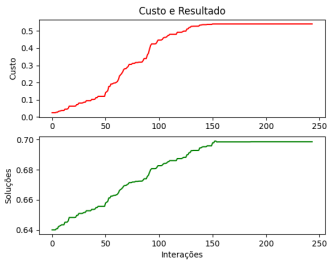


Algoritmo genético

Teste 3

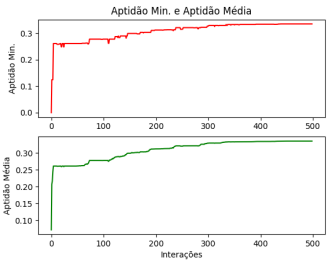
Aa Método	Solução (x)	Custo	Tempo
Hill Climbing	0.6986049315221323	0.5407978374347128	0.01486515998840332
Simulated Annealing	0.6985855984397057	0.5407975132858419	0.03974413871765137
Genetic Algorithm	0.8981096929855209	0.33527008184051704	0.24422192573547363

Semente: 1589469665.5963788



Hill Climbing

Simulated Annealing

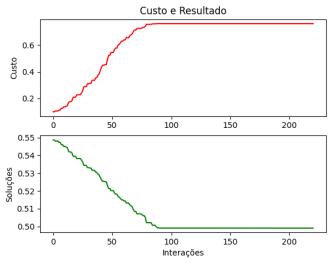


Algoritmo genético

Teste 4

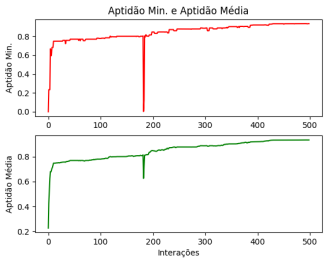
Aa Método	Solução (x)	Custo	Tempo
Hill Climbing	0.499081313325979	0.7609373295530815	0.010051727294921875
Simulated Annealing	0.4990718580791647	0.7609373213715259	0.09550976753234863
Genetic Algorithm	0.30040789708836146	0.9334520854537803	0.301668643951416

Semente: 1589469848.660967



Hill Climbing

Simulated Annealing

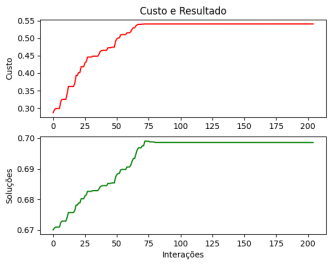


Algoritmo genético

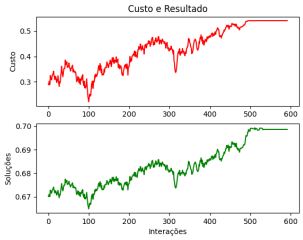
Teste 5

Aa Método	Solução (x)	Custo	Tempo
Hill Climbing	0.6986099064132327	0.5407978723018576	0.012197732925415039
Simulated Annealing	0.6985993241022688	0.5407977743139271	0.0651102066040039
Genetic Algorithm	0.49697891400197375	0.7584536166968291	0.1928386688232422

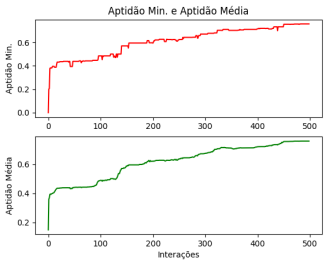
Semente: 1589470158.0770361



Hill Climbing



Simulated Annealing



Algoritmo genético

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

[-5, +5]

[-5, +5]

$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$

Para a resolução desse problema o algoritmo utilizado foi semelhante aos usados nas outras atividades, contando com algumas nuances. Em primeiro lugar, pode-se dizer que o algoritmo trabalhou com duas dimensões, podendo ser representado com uma matriz contendo dois vetores (para representar os indivíduos) ou com dois vetores separados, sendo a última abordagem escolhida para o desenvolvimento do algoritmo, dessa forma o primeiro vetor continha os indivíduos da população x e o segundo continha os indivíduos da população y.

Outro ponto importante é que esse algoritmo trabalhou com numeros representados em float, sendo possivel criar a primeira população dentro do intervalo de -5 a 5, que foi o dominio do problema.

Para a realização do crossover, assim como nas outras atividades, foi utilizado uma conversão para bitstring e aplicação de um corte aleatório que gerasse dois novos indivíduos. Já para a parte de mutação, dessa vez foi aplicado uma perturbação negativa, ou seja, sempre tirando um valor randômico do valor que fosse atual, na interação.

As condições de parada foram: repetição de 1/3 das interações máximas, numero máximo de interação atingido(500) e/ou ter individuo maior, ou menor que o domínio. Essas condições fazem com que o algoritmo não fique em um loop eterno.

Por fim, vale dizer que comumente nos testes, era possível visualizar uma descida grande na média das aptidões e depois uma alta elevação ou uma espécie de shoulder, em que a aptidão descia muito pouco até chegar no máximo de interações, além de que não era possível parar em 1/3 das repetições, pois o valor da aptidão continuava descendo, todavia infimamente.

Teste 1

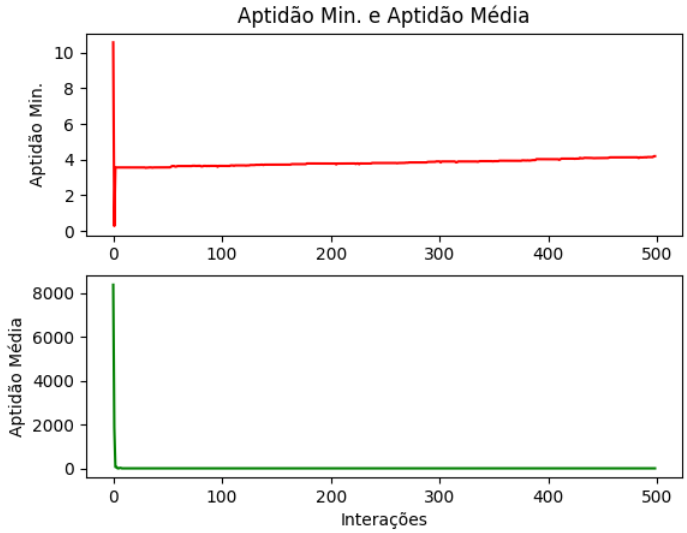
Semente: 1589946272.2241213

Tempo: 0.18619585037231445

Individuo X: -2.76

Individuo Y: 0.16

f(x,y): 5575.72



Teste 2

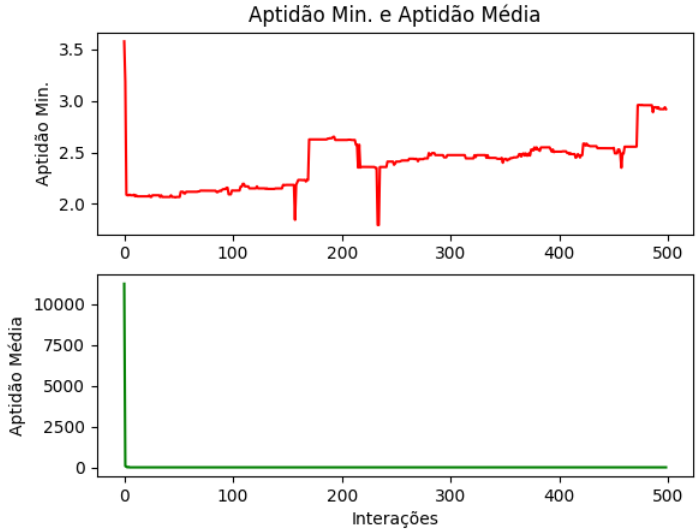
Semente: 1589946524.9781318

Tempo: 0.1061089038848877

Individuo X: 0.28

Individuo Y: 0.23

f(x,y): 2.94



Teste 3

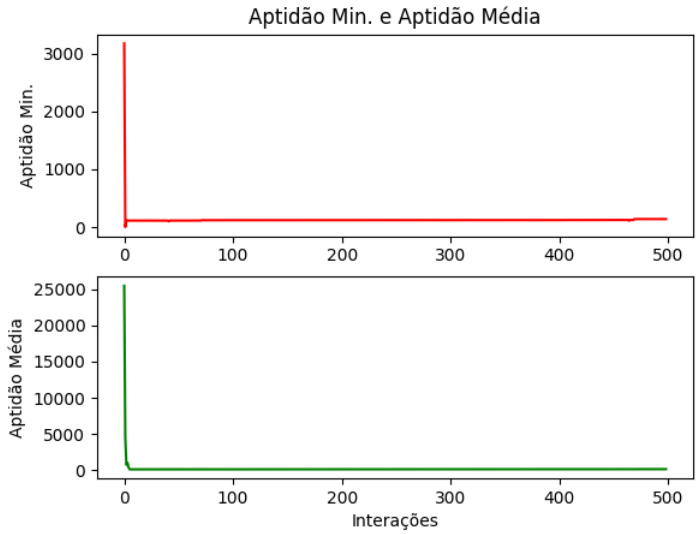
Semente: 1589946671.218997

Tempo: 0.10813331604003906

Individuo X: -2.96

Individuo Y: -3.24

f(x,y): 14419.52



Teste 4

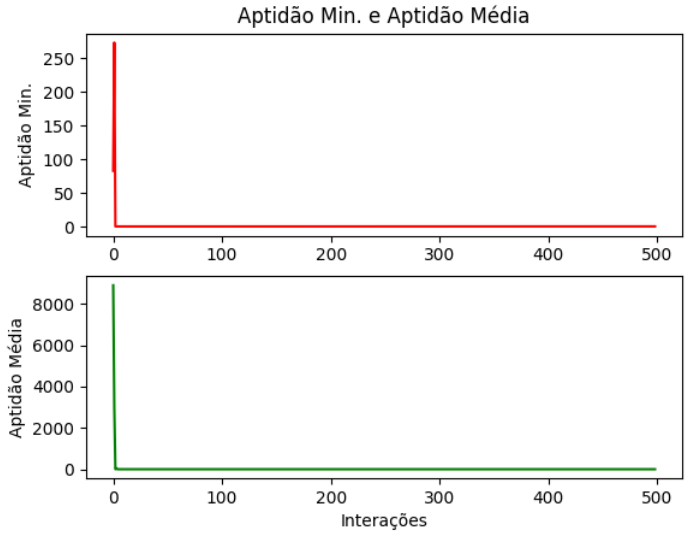
Semente: 1589946757.4502697

Tempo: 0.10974621772766113

Individuo X: 0.03

Individuo Y: 0.01

f(x,y): 0.95



Teste 5

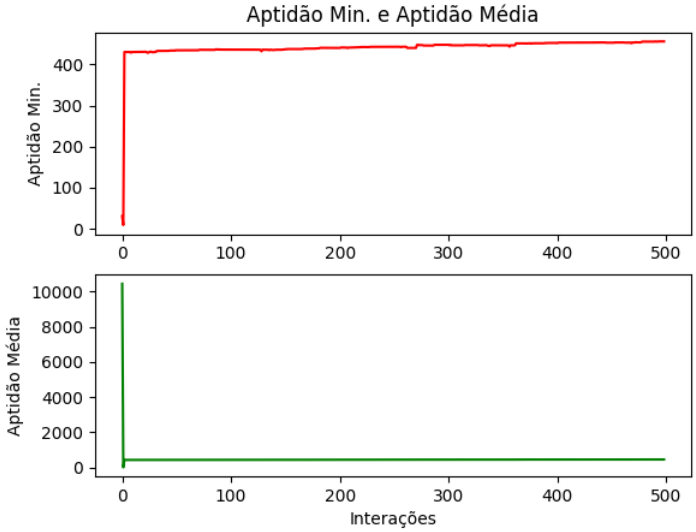
Semente: 1589946818.294662

Tempo: 0.10701894760131836

Individuo X: 0.99

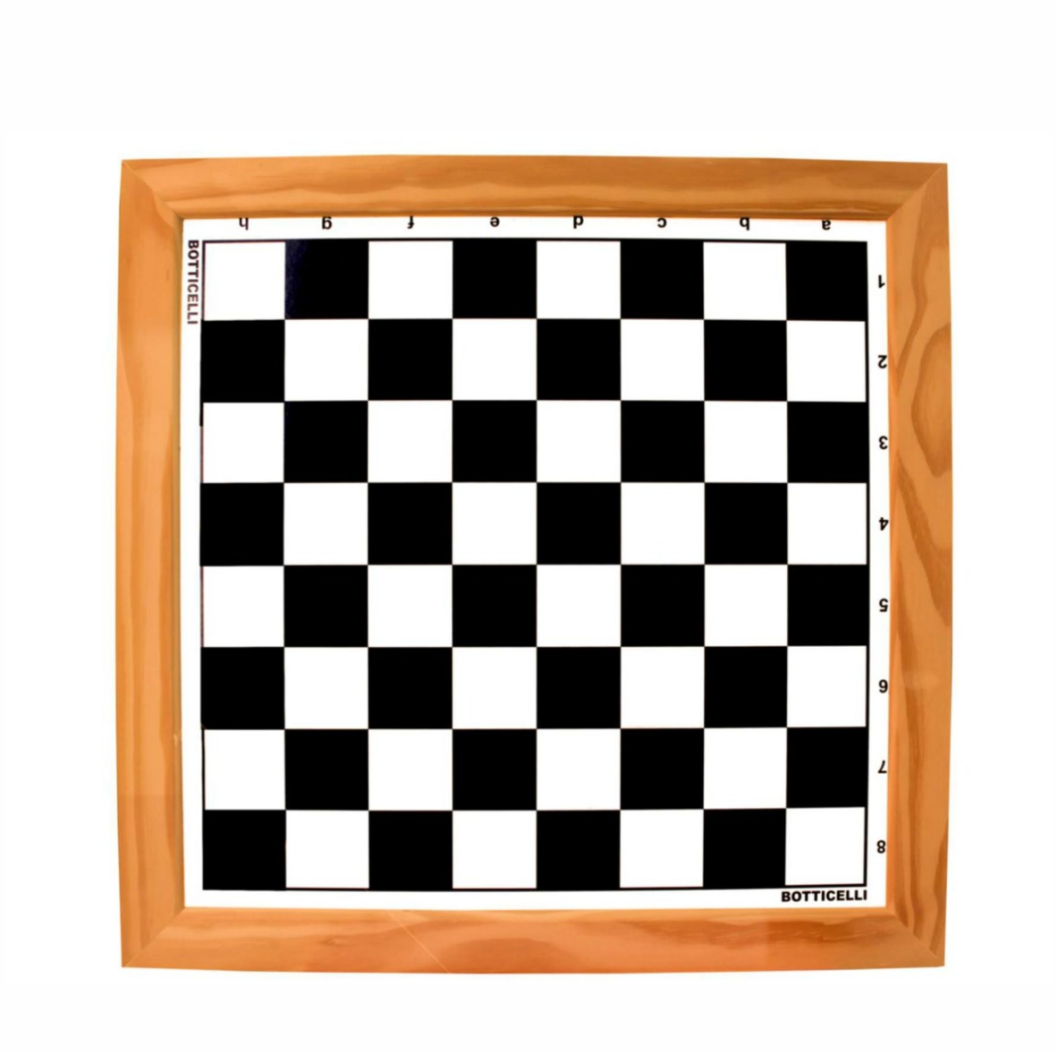
Individuo Y: -2.37

f(x,y): 1122.32



Adicional

Tendo em vista os algoritmos executados, foi pensado algo que pudesse representar objetos ou problemas do cotidiano, dessa forma o problema encontrado foi em montar um tabuleiro de damas, que é composto por uma malha com 8 quadrados verticais intercalando entre as cores pretas e brancas, além de 8 quadrados horizontais, também intercalados entre as cores pretas e bancas, como na imagem abaixo:



Para a execução desse código, a malha foi transformada em uma matriz, a qual foi utilizada para ser o objetivo do algoritmo, em que os quadrados brancos correspondiam a um bit 0 e os pretos correspondiam a um bit 1.

O algoritmo para o desenvolvimento foi semelhante ao execício 1, todavia, dessa vez o objetivo não era apenas uma bitstring e sim uma matriz por inteiro, dessa forma, para calcular a aptidão os indivíduos eram comparados com a linha que deveriam corresponder.

Além disso, os indivíduos eram compostos por 8 caracteres além de "0b" que é o padrão adotado pela linguagem Python para representar binários. Isso ocorreu para que fosse possível representar fidedignamente o tabuleiro de damas.

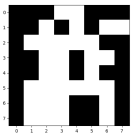
Todavia no primeiro teste percebeu-se que mesmo após 500.000 gerações o algoritmo não convergia, dessa forma, o parâmetro de hipótese de mutação foi alterado para 5% visto que a partir de testes observou-se que sempre havia uma

sequência de 1s ou 0s quando deveria haver uma intercalação e nunca uma sequência.

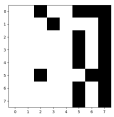
Após os testes com 5% de mutação (teste 2) também não se percebeu mudança significativa, apenas que os indivíduos caminhavam com uma aleatoriedade maior, dessa forma, diminuiu-se a taxa de mutação para 1.5% que é uma porcentagem dentro dos padrões, além de colocar um limitador para 2.000.000 de gerações e um corte fixo no crossover para a metade da bitstring.

Após o terceiro teste, em que ainda não havia obtido uma convergência, observou-se que a aptidão variava em uma média de 1 a 6, então foi decidido realizar o último teste utilizando mutação e deixando de usar crossover, visto que em alguns testes do exercício 1 o resultado convergia com mais eficácia, além de voltar a probabilidade para 2%, tentando deixar mais próximo dos testes no exercício 1.

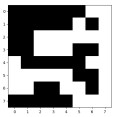
Teste 1



100.000 gerações

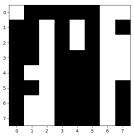


200.000 gerações



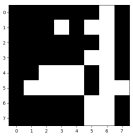
300.000 gerações

400.000 gerações

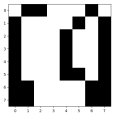


500.000 gerações

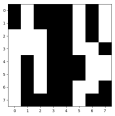
Teste 2



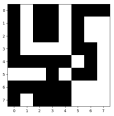
100.000 gerações



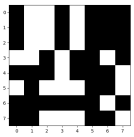
200.000 gerações



300.000 gerações

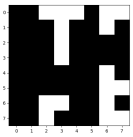


400.000 gerações

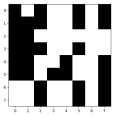


500.000 gerações

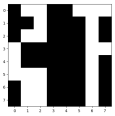
Teste 3



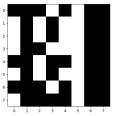
100.000 gerações



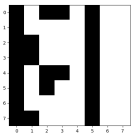
500.000 gerações



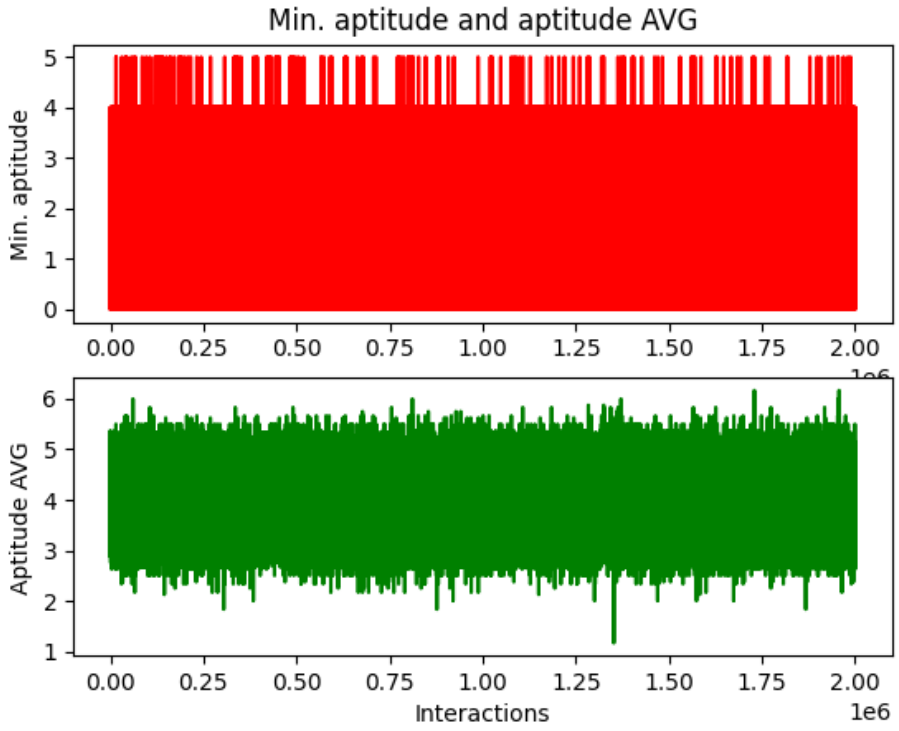
1.100.000 gerações



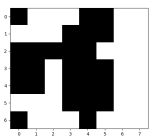
1.800.000 gerações



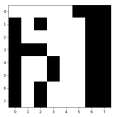
2.000.000 gerações



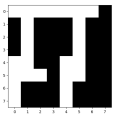
Teste 4



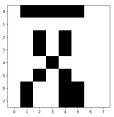
100.000 gerações



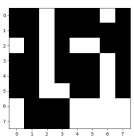
500.000 gerações



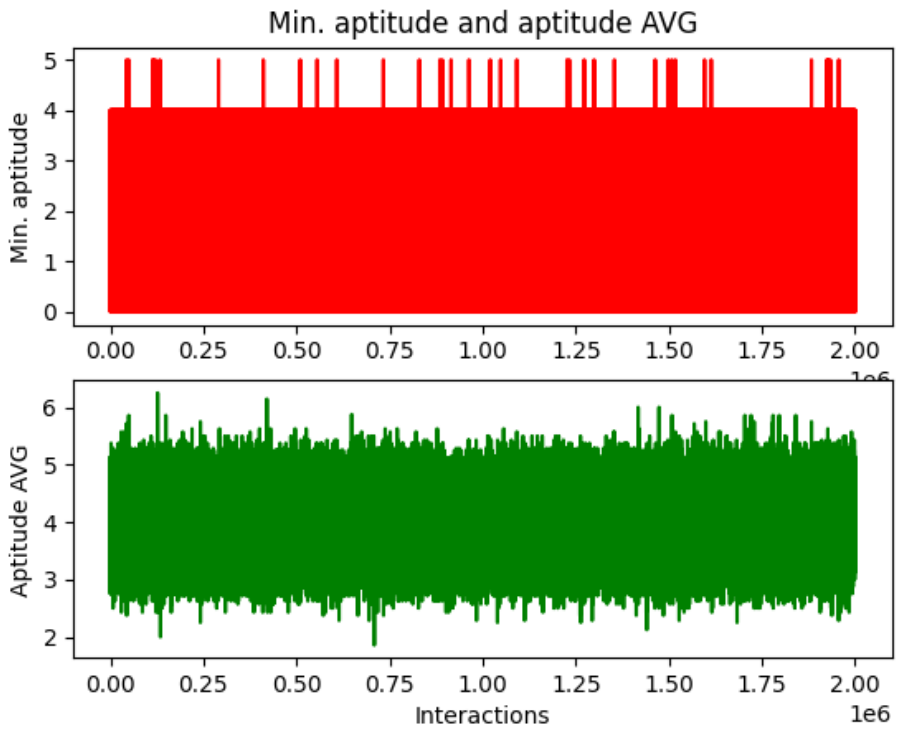
1.000.000 gerações



1.500.000 gerações



2.000.000 gerações



Conclusão sobre os adicionais

A partir dos testes não foi possível encontrar a matriz desejada, todavia por ser um algoritmo estocástico, há probabilidade de encontrar. Algumas considerações que devem ser relevadas para a falha de encontro da malha de dama com o algoritmo são:

A matriz que forma o tabuleiro de dama conta com duas variações de vetores, dessa forma era necessário que o algoritmo encontrasse dois padrões, dando assim uma dificuldade maior.

Além disso, vale considerar que um vetor que estivesse com uma aptidão boa para encontrar o primeiro padrão poderia realizar crossover com outro vetor que estivesse com uma aptidão boa para o segundo padrão, resultando em um novo vetor que seria piorado, provavelmente.

Por fim, diferente dos demais exercícios, em vez de encontrar um individuo que fosse igual ao objetivo, nesse teste era necessário que todos os indivíduos fossem perfeitamente iguais ao objetivo, tanto na questão de bits, quanto na questão de posicionamento, uma solução para esse problema seria estender os indivíduos da população, em que em vez de serem compostos por 8 bits, teriam de ser compostos por 64 bits, porém a representação visual dos mesmos seria totalmente diferente.