Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Graduação em Ciência da Computação
DCC191 - Computação Natural

Trabalho Prático 1

Aluno:
Gustavo P. Mitsuichi Oliveira

Setembro de 2016

Conteúdo

1 Introdução

O cubo mágico é um famoso brinquedo de puzzle que existe em todo o mundo. Se trata de um cubo com cada face subdividida em nove partes coloridas. O objetivo do cubo é mover cada face em sentido horário ou anti-horário até que todas subdivisões (facelets) de uma mesma face esteja com a mesma cor.

Figura 1: Cubos mágicos

O problema é que existem aproximadamente 519 quintilhões de estados possíveis desse cubo, tornando praticamente impossível de ser resolver utilizando métodos de força bruta. É sabido que já existem algoritmos eficientes para se resolver o cubo mágico, porém, nesse trabalho prático, a fim de exercitar os conceitos lecionados nas aulas de Computação Natural, foi sugerida uma abordagem utilizando algoritmos evolucionários.

Nesse trabalho, será descrito como foi o processo de implementação de um algoritmo e experimentação evolucionário para a resolução de um cubo mágico de dimensão 3x3.

2 Implementação

2.1 Processo de desenvolvimento

A linguagem escolhida para a implementação foi o Java Versão 8. Primeiramente foi necessário modelar o cubo mágico. Para isso criou-se uma matriz 6x3x3 do tipo String com cada elemento indicando a cor de um facelet. Na classe modelo do cubo foram criados métodos para a manipulação do cubo, realizando todos os movimentos possíveis.

A implementação do algoritmo evolucionário para a resolução do cubo começou com a tentativa de modelar o problema de maneira simples, sem a utilização de algoritmos conhecidos e operadores especiais. Nessa primeira tentativa, cada indivíduo era representado por uma string de movimentos. Cada pedaço dessa string (gene) era um movimento diferente, da seguinte forma: F, B, L, R, U, D, representavam giros em cada uma das faces no sentido horário, para o sentido anti-horário, se acrescentava o caractere 'i' (Fi, Bi...) e para giros duplos se acrescentava o caractere '2' (L2, R2...). O operador de mutação era aplicado ao indivíduo e cada um dos seus genes tinha uma certa probabilidade de ser trocado por algum gene aleatório, adicionado um gene vizinho a ele, ou ser removido, já que o indivíduo tem tamanho dinâmico. O operador de cruzamento, nesse caso, foi implementado baseando-se no cruzamento de dois pontos: eram selecionados aleatoriamente dois pontos do menor indivíduo e os genes entre esses dois pontos eram trocados com o outro indivíduo do cruzamento. A função de fitness contava quantos facelets (subdivisões de uma face do cubo) estavam na posição correta. Para tando, era comparada a cor de cada facelet com a cor do facelet central. A fitness poderia assumir os valore de 6 a 54 (cubo resolvido).

Esse primeiro algoritmo não foi uma boa escolha já que sempre chegava a um ótimo local, gerando indivíduos com fitness por volta de 32, com a maioria das combinações de parâmetro inicial. Suspeitou-se que o problema do algortimo estava na função fitness, já que dessa forma, um indivíduo com uma fitness maior poderia estar mais longe de ser resolvido que um indivíduo com uma fitness menor. Isso levou a implementação de um segundo algoritmo, dessa vez utilizando elementos do algoritmo de resolução do cubo 3x3 [2].

A diferença para o algoritmo anterior é que agora cada indivíduo possui um atributo para indicar que até certo gene corresponde a um determinado passo do algoritmo, ou seja, o indivíduo está correto até certo ponto. Os operadores de mutação e cruzamento são semelhantes ao do algoritmo anterior porém, o indivíduo só é alterado a partir do ponto anteriormente explicado. A função de fitness, além de considerar se cada facelet está correto, como no

algoritmo anterior, verifica a cada avaliação de gene se o cubo está em algum passo do algoritmo, se estiver, a função atribui mais 20 pontos por cada passo. Essa fitness tem como valores possíveis 6 a 154, já que são considerados 5 passos do algoritmo de resolução.

Como no primeiro algoritmo, esse também não foi uma boa escolha. O melhor indvíduo observado chegou apenas até o primeiro passo do algoritmo de resolução e a estratégia não apresentava bons resultados com a variação de parâmetros: a média e o desvio padrão não aumentavam como esperado, o cruzamento não gerava bons filhos. a suspeita dessa vez foi de que o problema estava na representação do indivíduo, já que, pequenas mudanças em indivíduo gerava um outro completamente diferente, o que prejudicava a exploração espacial. O insucesso nessa estratégia levou a implementação do terceiro algoritmo, que será descrito com mais detalhes na próxima subseção.

2.2 A solução final

O terceiro e último algoritmo implementado se baseia na solução proposta por Herdy et al. [3] e avaliado por El-Sourani et al. [1]. Nessa estratégia, cada gene do indivíduo é um conjunto de movimentos que altera pouco na configuração final do cubo. A relação entre gene e movimentos está descrita na tabela 1.

Gene	Movimentos
1	FRBLULiUBiRiFiLiUiLUi
2	FiLiBiRiUiRUiBLFRURiU
3	LDiLiFiDiFUFiDFLDLiUi
4	RiDRFDFiUiFDiFiRiDiRU
5	UF2UiRiDiLiF2LDR
6	UiF2ULDRF2RiDiLi
7	RiURUiRiUFRBiRBRFiR2
8	LUiLiULUiFiLiBLiBiLiFL2
9	FiUBUiFUBiUi
10	FUiBiUFiUiBU
11	RLiU2RiLF2
12	LiRU2LRiF2

Tabela 1: Tabela de Movimentos

O algoritmo começa com a leitura do arquivo de entrada, conforme o padrão do enunciado do trabalho prático. Feita a leitura o cubo é inicializado e a população inicial é gerada, de forma totalmente aleatória. Em seguida a fitness de todos os indivíduos da população é calculada da seguinte forma:

1. Cada gene do indivíduo é decodificado conforme a tabela 1 e os movimentos correspondentes são aplicados à uma cópia do cubo original em cada uma das 6 possíveis orientações, a maior fitness será armazenada.

2. São contados quantos facelets estão na posição correta, desconsiderando os centrais, e esse número é adicionado à fitness

3. São contados quantos cantos (corners) do cubo estão na posição correta e esse número multiplicado por 6 é adicionado à fitness.

4. São contados quantas extremidades (edges) do cubo estão na posição correta e esse número multiplicado por 4 é adicionado à fitness.

A fitness calculada dessa forma pode ir de 0 a 144.

Calculada a fitness da população inicial, o algoritmo entra no loop principal de execução. Os indivíduos da população são submetidos à uma seleção por torneio e em seguida são aplicados os operadores de cruzamento (com certa probabilidade) e mutação nos indivíduos selecionados.

1. **Cruzamento** São selecionados aleatoriamente dois pontos do individuo de menor tamanho para que os genes entre esses pontos sejam substituídos pelos genes entre os mesmos dois pontos do maior indivíduo. Dois filhos são gerados.

2. **Mutação** Cada gene do indivíduo tem uma certa probabilidade de sofrer um dos três tipos de mutação. O primeiro é a mudança do gene por outro diferente, gerado aleatoriamente. O segundo é a adição de um novo gene logo em seguida do gene atual. O terceiro tipo é a remoção do gene.

O algoritmo seleciona e aplica os operadores aos indivíduos até que seja gerado um determinado número para completar a próxima população. O algoritmo termina quando se completa um determinado número de gerações.

2.3 O código

O código do programa que implementa o algoritmo está divido em três classes:

- **Cube.java** : É onde se encontra a modelagem do cubo. A leitura do arquivo de entrada é feita em um construtor da classe que recebe o caminho por parâmetro. Os métodos de movimentação do cubo também foram implementados aqui. O método **makeMomement** é responsável por receber um gene (1,2...12) e convertê-lo para os movimentos do cubo dependendo de qual face será a face frontal.

- **Individual.java** : Representa o indivíduo do algoritmo genético. Aqui estão Array de Strings representando os genes e também o valor da fitness para o indivíduo. Também existe um método estático que gera um indivíduo aleatório.
- **MainClass.java** : É a classe principal do programa. No método main se encontra o fluxo principal, onde é descrito o que é feito a cada geração. Na classe também se encontram os métodos responsáveis pelos operadores de seleção por torneio, cruzamento e mutação. Todos os parâmetros utilizados nos experimentos estão no topo dessa classe.

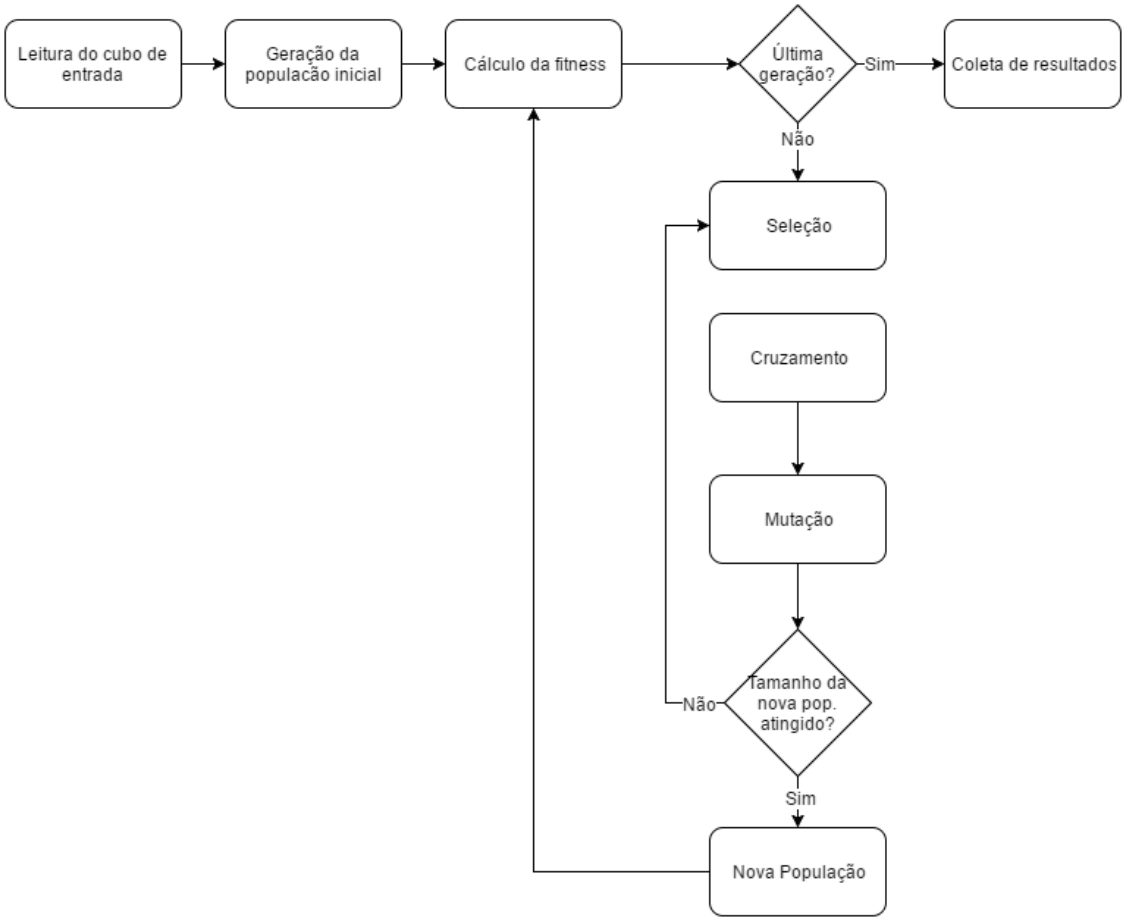

Figura 2: Diagrama de representação do algoritmo

3 Experimentos

Nessa seção serão descritos os principais experimentos realizados com o algoritmo previamente descrito.

3.1 Experimento 1 - Inicial

- População: 300
- Gerações: 300
- Pc: 0.9
- Pm: 0.05
- Torneio: 2
- Elitismo: sim

Nesse primeiro experimento, o objetivo foi de verificar como o algoritmo se comportava, com valores iniciais. Para cada uma das 3 entradas foram feitos 30 testes e computados os valores médios. O gráfico que mostra as gerações leva em conta a melhor execução para cada um das entradas. A tabela 2 mostra a média de cada um dos valores coletados para cada uma das execuções

Entrada	Média	Desv. padrão	Melhor ind.	Pior ind.	Ind. iguais
in1	34,67	8,33	49,72	14,34	155,97
in2	35,88	8,63	53,95	15,30	159,50
in3	28,20	8,19	45,50	10,00	150,00

Tabela 2: Experimento 1 - Valores médio para as entradas

É possível notar que o algoritmo não chega na melhor solução (144) com os valores propostos. É interessante destacar também que metade da população é idêntica, o que leva a pensar que o problema pode estar na variedade da população.

Um outro dado interessante, mostrado na tabela 3, é que os cruzamentos não geram bons indivíduos em sua maioria. Usar um Pc menor, pode ajudar na qualidade geral da população.

As figuras 3, 4 e 5 mostram como foram as execuções para cada uma das entradas. Nota-se que houve uma convergência muito rápida para um ótimo local, o que prejudicou a solução final.

Entrada	Filhos piores	Filhos melhores
in1	211,24	57,79
in2	214,10	56,00
in3	192,00	79,00

Tabela 3: Experimento 1 - Qualidade do cruzamento

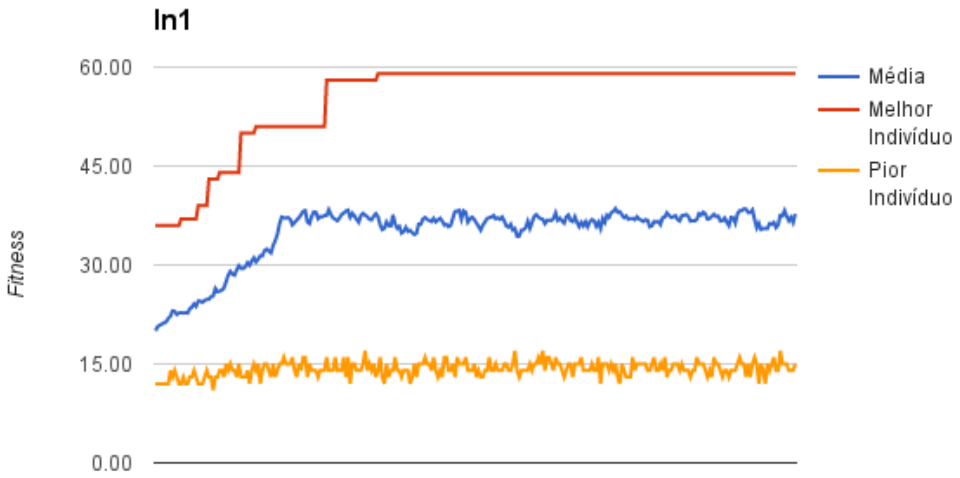

Figura 3: Gráfico de execução para in1

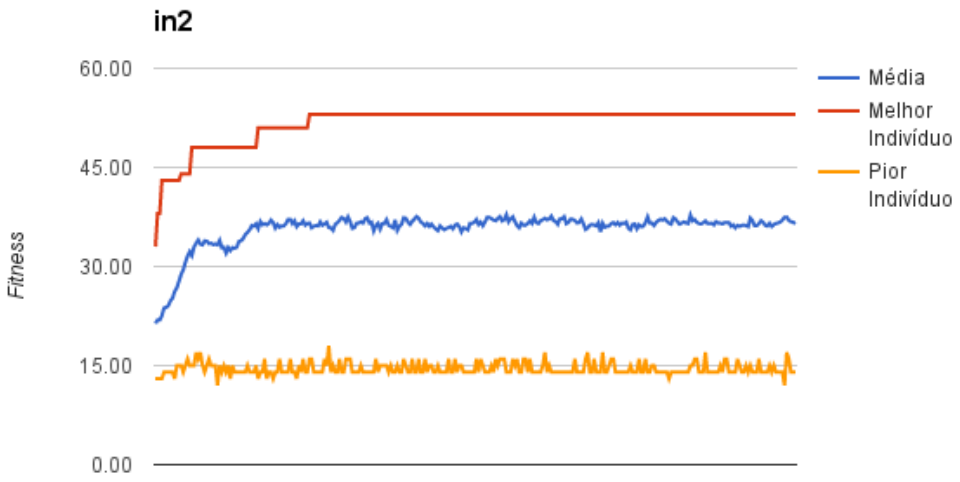

Figura 4: Gráfico de execução para in2

Figura 5: Gráfico de execução para in3

3.2 Experimento 2 - Pc e Pm

- População: 300
- Gerações: 300
- Pc: 0.6
- Pm: 0.3
- Torneio: 2
- Elitismo: sim

Nesse experimento, foram modificados os parâmetros de probabilidade de mutação e cruzamento. Novamente foram executados 30 experimentos para cada entrada e a tabela 4 computa as médias para cada entrada.

Entrada	Média	Desvio padrão	Melhor ind.	Pior ind.	Indivíduos iguais
in1	20,05	4,10	43,87	12,40	1,80
in2	21,01	4,47	47,67	12,93	1,93
in3	15,20	4,03	40,20	8,40	0,13

Tabela 4: Experimento 2 - Valores médio para as entradas

Analisando a tabela é possível verificar que o resultado foi pior que o anterior, mesmo com menos indivíduos iguais a cada geração. Isso se deve ao fato que o melhor indivíduo não foi muito bem explorado na evolução. O operador de cruzamento se mostra mais eficiente que o de mutação que deve ser usado com cautela.

As figuras 6, 7 e 8 mostram o resultado de se utilizar uma taxa grande de mutação. Nota-se que não houve uma boa convergência já que a média e o pior indivíduo se mantêm quase constante com o passar das gerações, mesmo com o melhor indivíduo melhorando um pouco.

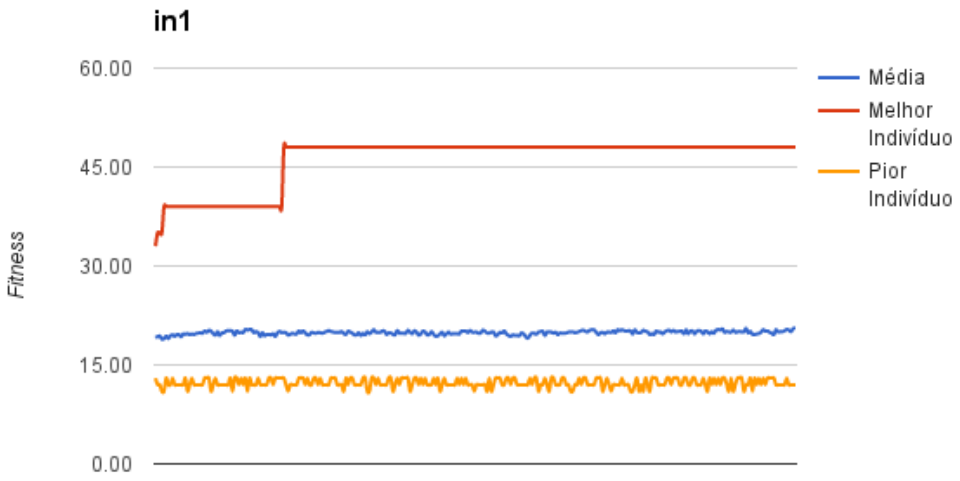

Figura 6: Gráfico de execução para in1

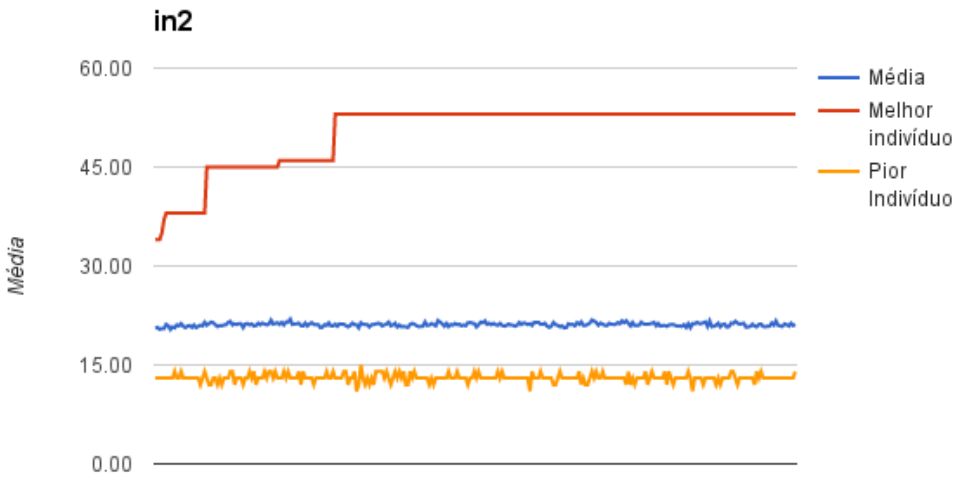

Figura 7: Gráfico de execução para in2

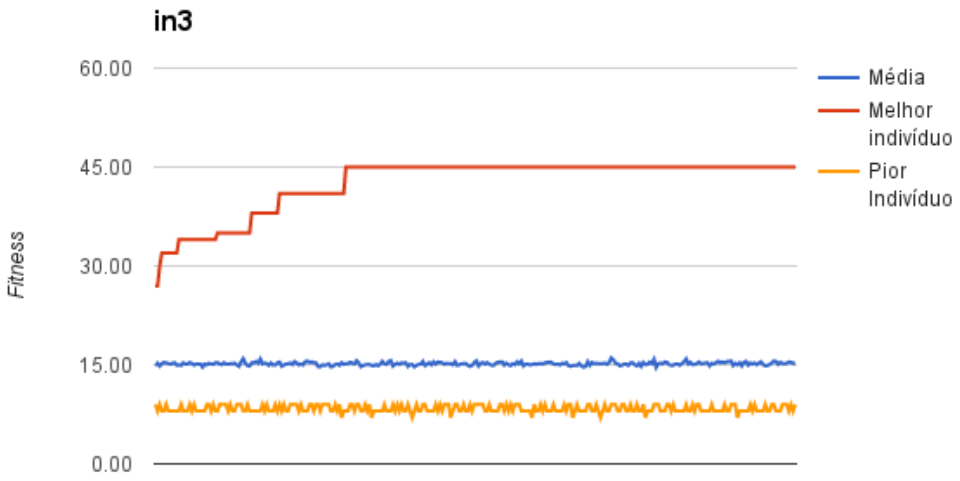

Figura 8: Gráfico de execução para in3

3.3 Experimento 3 - Torneio

- População: 1000
- Gerações: 500
- Pc: 0.6
- Pm: 0.3
- Torneio: 10
- Elitismo: sim

Esse experimento foi feito com o intuito de testar um novo número de população a fim de aumentar a variedade e melhorar a solução encontrada e consequentemente ajustando o tamanho do torneio para que seja uma seleção mais adequada para o tamanho da solução. O resultado médio está compilado na tabela 5.

É possível ver como o aumento da população aumentou cerca de 50% as soluções encontradas para todas as entrada. A solução ótima ainda não foi encontrada com os parâmetros atuais e a convergência ainda foi precoce. As

Entrada	Média	Desvio padrão	Melhor ind.	Pior ind.	Indivíduos iguais
in1	39,90	12,20	59,20	13,40	223,95
in2	41,56	13,86	61,05	14,25	415,80
in3	34,91	13,73	55,00	9,41	357,59

Tabela 5: Experimento 3 - Valores médio para as entradas

figuras 9, 10 e 11 mostram como evoluíram os valores para cada entrada. Destaque para a entrada in2 que, com o conjunto de parâmetros do experimento, obteve o melhor resultado entre todos os experimentos.

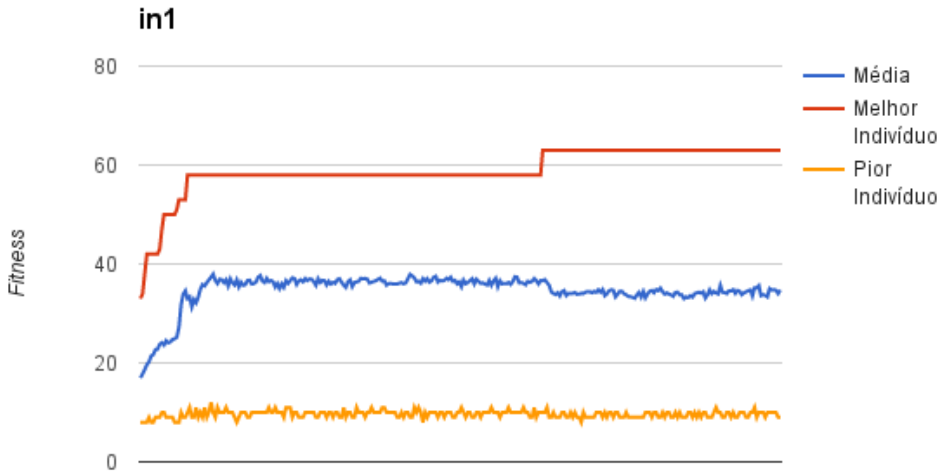

Figura 9: Gráfico de execução para in1

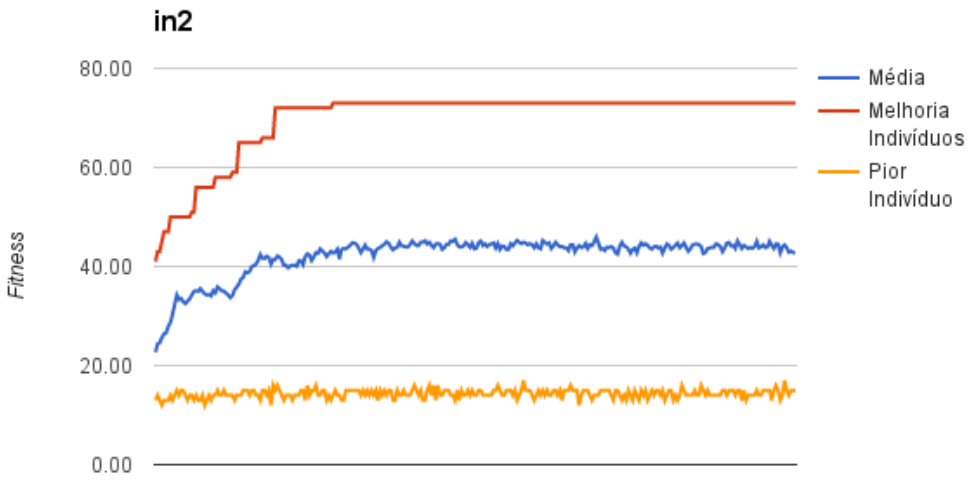

Figura 10: Gráfico de execução para in2

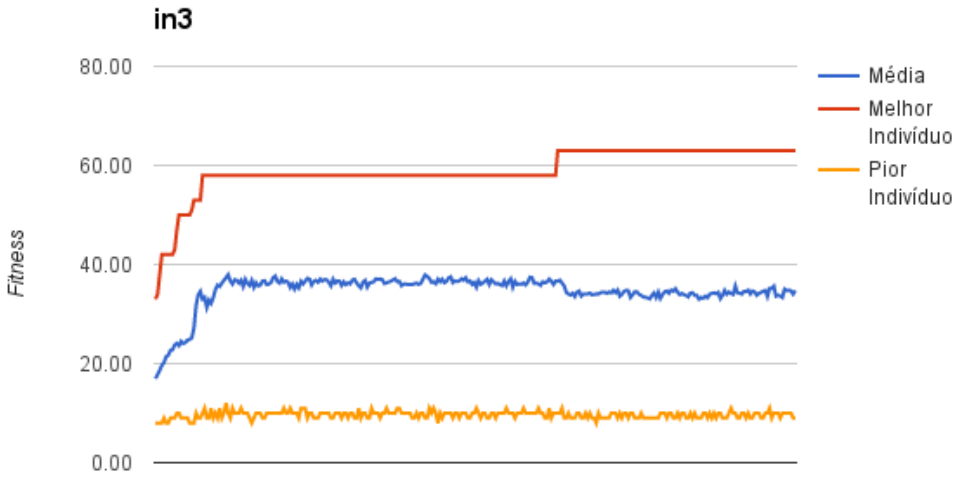

Figura 11: Gráfico de execução para in3

3.4 Experimento 4 - Elitismo

- População: 1000
- Gerações: 500
- Pc: 0.6
- Pm: 0.3
- Torneio: 2
- Elitismo: não

Os experimentos sem o elitismo sempre pioraram os melhores indivíduos e demais valores médios das execuções. Os dados não foram relevantes e estavam muito dispersos, portanto considerou-se os resultados sem relevância e eles não serão apresentados.

4 Conclusão

Nesse trabalho prático, houve um insucesso em encontrar a solução para o cubo mágico 3x3 utilizando uma estratégia evolucionária. Houve um grande esforço para desenvolver uma abordagem que gerasse resultados satisfatórios como pode ser observado na seção 2. O fato de que nem mesmo a solução da literatura[3] gerou a solução ótima, leva a pensar que o erro pode estar na modelagem do cubo mágico que é um tanto complexo de se implementar.

Apesar de não ter sido encontrada a solução ótima, houve um grande aprendizado devido à implementação do algoritmo genético e avaliação dos experimentos, sem contar com os artigos que foram lidos sobre a utilização de estratégias evolucionárias na resolução do cubo mágico e sobre o cubo mágico somente [2] .

Referências

[1] Nail El-Sourani e Markus Borschbach. "Design and comparison of two evolutionary approaches for solving the rubik's cube". Em: *International Conference on Parallel Problem Solving from Nature*. Springer. 2010, pp. 442–451.

[2] Dénes Ferenc. *How to solve the Rubik's Cube*. URL: `https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/`.

[3] M Herdy e G Patone. "Evolution Strategy in Action, 10 ES-Demonstrations". Em: *Proceedings of the International Conference on Evolutionary Computation: The Third Parallel Problem Solving from Nature (PPSN III)*. 1994, pp. 1–17.