A Aplicação de Algoritmos Genéticos para a Resolução do Cubo de Rubik

Bruno L. Sousa[1]

[1]Instituto de Ciências Exatas – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

bruno.luan.sousa@gmail.com

Resumo. *Este trabalho tem como objetivo avaliar, discutir e analisar o problema de resolver o cubo mágico e a aplicação dos algoritmos genéticos para resolver esse problema. Baseado nisso, foi realizado uma modelagem do problema em algoritmos genéticos e implementado uma estrutura de dados junto ao algoritmo. Ainda, como forma de validar a eficiência desse algoritmo na resolução desse problema, foram realizados vários testes e experimentos, dos quais foi possível perceber que a solução possui uma evolução durante a execução do algoritmo, porém nem sempre o algoritmo genético consegue efetuar a resolução do cubo.*

1. Introdução

O cubo de Rubik, também conhecido como cubo mágico, é um quebra-cabeça tridimensional no qual utiliza-se uma sequência de movimentos lógicos para poder resolvê-lo. A versão padrão do cubo (3x3), mostrada pela **Figura 1**, é composta por 26 subcubos, dos quais 6 compõe o centro, 12 compõe as bordas e 8 compõe as quinas. Cada parte visível do cubo é rotulada por uma cor e cada movimento altera a posição de quatro quinas e quatro bordas. Assim, a meta é realizar pequenos movimentos no cubo, a partir de um estado inicial, a fim de agrupar todos os rótulos de mesma cor em uma mesma face. Porém, resolver o cubo mágico não é trivial, uma vez que, exitem 4,3 10^{19} estados diferentes que podem ser atingidos a partir de qualquer configuração dada.

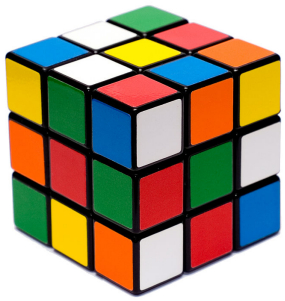

Figura 1. Cubo mágico padrão.

O objetivo desse trabalho é utilizar o algoritmo genético para resolver o cubo, a partir de um dado estado. Com isso, busca-se uma solução que aproxime-se da solução ótima por meio da evolução de m população, nas quais estão representadas possíveis soluções do espaço de busca do problema.

A seção 2 deste trabalho discute a modelagem do cubo mágico para um algoritmo genética, destacando pontos como: a representação dos indivíduos, a representação dos operadores, dentre outros. A seção 3 exibe as decisões de implementação, bem como a representação das classes e demais estruturas utilizadas para implementar a resolução deste problema. A seção 4 informa algumas instruções para execução do algoritmo implementado. A seção 5 apresenta os experimentos realizados e a metodologia adotada. A seção 6 apresenta os resultados e análises dos experimentos. Por fim, a seção 6 conclui este trabalho.

2. Modelagem do Cubo Mágico

A modelagem do cubo mágico em uma solução baseada no algoritmo genético foi dividida em três fases: modelagem do indivíduo, escolha de uma função de avaliação (Fitness), definição dos operadores de cruzamento e mutação utilizados para criação de uma nova população, e apresentação do algoritmo proposto. Cada uma dessas fases é apresentada a seguir.

2.1. Modelagem do Indivíduo

Em qualquer problema que utiliza algoritmos genéticos como possível solução é necessário realizar a formulação de duas características essenciais desses problema: o genótipo, fator que define a composição das características de um indivíduo, e o fenótipo, parte que permite visualização das características de um indivíduo, definidas pelo genótipo.

Nesse problema, um indivíduo representa uma solução válida para a resolução do cubo, ou seja, uma sequência de movimentos que a partir de um dado estado do cubo, consiga encaixar cada uma das cores em uma respectiva face, resolvendo assim o cubo. Essa sequência de movimentos que compõe um indivíduo constitui o genótipo. Porém, apenas as sequências de movimentos por si só, não permite que um usuário consiga visualizar a resolução do cubo. Assim, é necessário que cada indivíduo possua a configuração inicial do cubo, para que se possa visualizar o resultado da aplicação do genótipo e analisar a qualidade da solução. Essa representação do cubo compõe o fenótipo de um indivíduo.

Um primeiro passo na representação de um indivíduo foi definir quais seriam os movimentos que formariam o seu genótipo. Para isso, foram criados dois tipos de movimentos: os movimentos considerados primários e os movimentos considerados compostos.

Os movimentos primários constituem 18 rotações simples das faces dos cubos, identificados por um caractere, que podem comportar-se de três formas diferentes: 90° sentido horário, 90° sentido anti-horário e 180° sentido horário. Os movimentos estão especificados na **Tabela 1**, junto com sua respectiva ação.

Os movimentos compostos são sequências de movimentos primários agrupados, os quais ao serem executados realizam pequenas alterações no cubo. Esses movimentos foram propostos por [Herdy and Patone 1994] e foram incorporados a este trabalho como forma de manter uma semântica na alteração do cubo, uma vez que os movimentos primários, por si só, poderiam deixar as soluções muito aleatórias, e em função disso, gerar soluções ruins. Os movimentos compostos estão especificados na **Tabela 2**.

Tabela 1. Descrição dos movimentos primários e suas respectivas ações.

Movimento	Ação
B	Rotaciona a face do fundo do cubo 90º em sentido horário.
D	Rotaciona a face de baixo do cubo 90º em sentido horário.
F	Rotaciona a face da frente do cubo 90º em sentido horário.
L	Rotaciona a face da esquerda do cubo 90º em sentido horário.
R	Rotaciona a face da direita do cubo 90º em sentido horário.
U	Rotaciona a face de cima do cubo 90º em sentido horário.
B1	Rotaciona a face do fundo do cubo 90º em sentido anti-horário.
D1	Rotaciona a face de baixo do cubo 90º em sentido anti-horário.
F1	Rotaciona a face da frente do cubo 90º em sentido anti-horário.
L1	Rotaciona a face da esquerda do cubo 90º em sentido anti-horário.
R1	Rotaciona a face da direita do cubo 90º em sentido anti-horário.
U1	Rotaciona a face de cima do cubo 90º em sentido anti-horário.
B2	Rotaciona a face do fundo do cubo 180º em sentido horário.
D2	Rotaciona a face de baixo do cubo 180º em sentido horário.
F2	Rotaciona a face da frente do cubo 180º em sentido horário.
L2	Rotaciona a face da esquerda do cubo 180º em sentido horário.
R2	Rotaciona a face da direita do cubo 180º em sentido horário.
U2	Rotaciona a face de cima do cubo 180º em sentido horário.

Tabela 2. Descrição dos movimentos compostos.

Movimento	**Nome do Movimento**	**Sequência de Movimentos Primários Correspondentes**
TEFCW	Two Edge Flip CW	F R B L U L1 U B1 R1 F1 L1 U1 L,U1
TEFCCW	Two Edge Flip CCW	F1 L1 B1 R1 U1 R U1 B L F R U,R1 U
TCFCW	Two Corner Flip CW	L D1 L1 F1 D1 F U F1 D F L D L1,U1
TCFCCW	Two Corner Flip CCW	R1 D R F D F1 U1 F D1 F1 R1 D1 R U
TESCW	Three Edge Swap CW	U F2 U1 R1 D1 L1 F2 L D R
TESCCW	Three Edge Swap CCW	U1 F2 U L D R F2 R1 D1 L1
TECSCW	Two Edge/Corner Swap CW	R1 U R U1 R1 U F R B1 R B R F1 R2
TECSCCW	Two Edge/Corner Swap CCW	L U1 L1 U L U1 F1 L1 B L1 B1 L1 F L2
TCSCW	Three Corner Swap CW	F1 U B U1 F U B1 U1
TCSCCW	Three Corner Swap CCW	F U1 B1 U F1 U1 B U
TIESCW	Three Inslice Edge Swap CW	R L1 U2 R1 L F2
TIESCCW	Three Inslice Edge Swap CCW	L1 R U2 L R1 F2

A partir desses dois grupos de movimentos é possível representar soluções para a resolução do Cubo mágico. Neste trabalho, foi definido um tamanho máximo de 30 genes, ou seja, 30 sequências de movimentos para cada um dos cromossomos existentes na população, que estão representados em forma de um vetor. Além desta representação ser simples, ela facilita a realização das operações de cruzamentos e mutações, cujas estão descritas na seção 2. 3 e 2.4 respectivamente.

O fenótipo referente a cada indivíduo é representado pelo próprio cubo mágico.

Quando um novo cromossomo é criado, por padrão é atribuído a configuração inicial do cubo que se deseja encontrar uma solução. Após atribuir os movimentos no genótipo de cada indivíduo, os movimentos são executados, e o fenótipo é atualizado de acordo com cada nova configuração sofrida pelo cubo, em decorrência da aplicação dos movimentos. Ao final desses movimentos, a aptidão deste indivíduo é calculada por meio dessa configuração final do cubo. Para uma maior facilidade na execução dos movimentos, o fenótipo está representado por uma lista de matrizes 3×3, onde cada matriz representa a configuração de uma face existente no cubo, e são armazenadas em uma lista. A **Figura 2** ilustra essa representação do fenótipo.

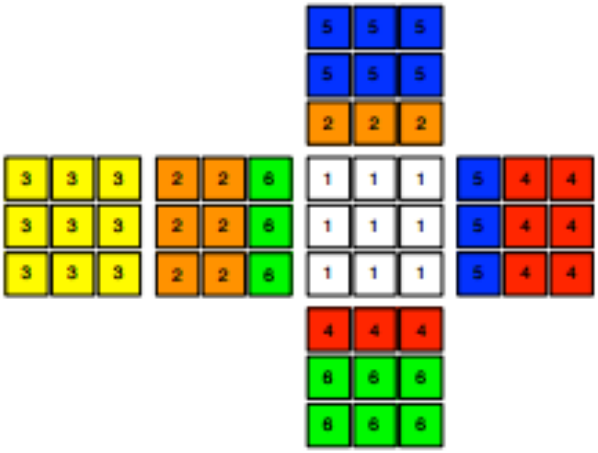

Figura 2. Fenótipo utilizado para cada indivíduo.

2.2. Função Fitness

A função Fitness utilizada neste trabalho é dada pela soma de penalidades atribuídas a alguns aspectos posicionados de forma errada, cujos quais influenciam no resultado final do cubo. Neste trabalho está sendo utilizado três tipos de penalidades diferentes. A primeira penalidade tem por objetivo penalizar os quadrados de cada face que estão posicionados errados. A orientação da cor de cada face é dada pelo quadrado central. Essa primeira penalidade tem peso um e multiplica esse peso pela quantidade total de quadrados posicionados de forma errada no cubo.

A segunda penalidade tem por objetivo penalizar bordas que estão incorretamente posicionadas. Uma borda está corretamente posicionada quando existe um quadrado que liga a borda ao centro do cubo. De uma forma mais simples, o posicionamento de uma borda é considerado correto, quando esta possui em uma de suas respectivas faces o quadrado situado no meio da bor com a mesma cor que o quadrado situado no centro da face. A penalidade para posicionamento errado de bordas possui peso 4. Como um cubo possui 12 bordas, o máximo que essa penalidade pode atingir é 48.

A terceira penalidade tem por objetivo penalizar as quinas do cubo que estão incorretamente posicionadas. Uma quina está corretamente posicionada quando suas três são iguais as cores que estão posicionadas no centro de cada uma das faces que a mesma tem acesso. Essa penalidade possui peso 6, e como um cubo possui 8 quinas, o máximo que essa punição pode atingir é 48 também.

De acordo com essa Fitness, a penalidade máxima que um cubo pode possui é 144, uma vez que cada punição isolada atinge um máximo de 48. Assim, como são três tipos diferentes, $3 \times 48 = 144$.

2.3. Cruzamento e Mutação

Cruzamento e Mutação são dois tipos de operadores essenciais em um algoritmo genético, pois são eles os responsáveis pela evolução das soluções ao longo das gerações. Cada um desses operadores é descrito separadamente a seguir, iniciando pelo operador de cruzamento.

2.3.1. Cruzamento

Para aplicar o operador de cruzamento sempre é necessário a escolha de pais, os quais originam dois filhos, por meio de uma permutação de genes entre os dois ancestrais. A forma com que a permutação é realizada, varia de algoritmo para algoritmo. Neste trabalho, o cruzamento baseou-se na estratégia de ponto de corte. Essa estratégia consiste em escolher aleatoriamente uma posição do vetor em que o genótipo está situado, e esta é utilizada como o ponto limite para a permutação. O método de seleção utilizada para escolha dos pais candidatos a cruzarem é realizada por um torneio.

A seguir, é mostrado um exemplo de estratégia de cruzamento por meio de ponto de corte. Para um melhor entendimento, considere para a ilustração a seguir, um vetor de tamanho 10 e com um índice de ponto de corte, selecionado aleatoriamente, igual a 4.

$$\text{ind1}\{genotipo = [F, B, D, F1|U, L, B2, F2, R, U2]\}$$
$$\text{ind2}\{genotipo = [D, U1, F, U|B1, U, F2, L, R, U1]\}$$

Após a permutação dos pais, tem-se:

$$\text{ind3}\{genotipo = [F, B, D, F1|B1, U, F2, L, R, U1]\}$$
$$\text{ind4}\{genotipo = [D, U1, F, U|U, L, B2, F2, R, U2]\}$$

2.3.2. Mutação

A mutação é um tipo de operador utilizado para proporcionar diversidade em uma população. A mutação utilizada para diversificar os indivíduos foi a mutação uniforme. Para este tipo de mutação é verificado a probabilidade de cada gene pertencente a um cromossomo sofrer mutação. O gene apenas sofre mutação se ele for selecionado. Quando isso acontece, é selecionado um outro gene de forma aleatória e substituído pelo atual.

A seguir, é mostrado um exemplo de estratégia de mutação uniforme. Para um melhor entendimento, considere para a ilustração a seguir, um vetor de tamanho 10 e que os índices: 4, 6 e 9, foram selecionados para sofrer mutação.

$$\text{ind1}\{genotipo = [F, B, D, F, \mathbf{U}, L, \mathbf{B2}, F2, R, \mathbf{U2}]\}$$

Após a ocorrência da mutação, tem-se:

$$\text{ind2}\{genotipo = [F, B, D, F, \mathbf{R}, L, \mathbf{D1}, F2, R, \mathbf{B}]\}$$

2.4. Algoritmo Genético

O algoritmo proposto é relativamente bem simples e seu funcionamento pode ser resumido nos seguintes passos ilustrados na **Figura 8**.

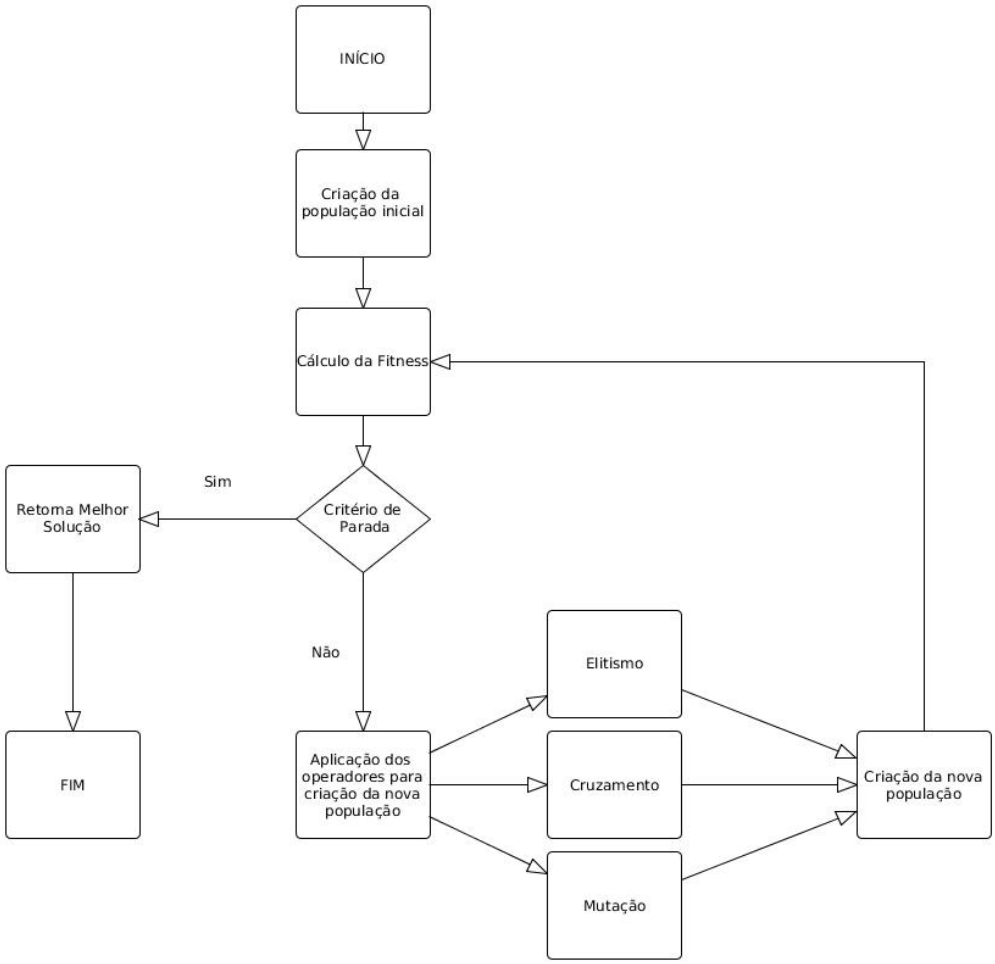

Figura 3. Fluxograma do passo a passo do algoritmo genético.

O algoritmo é bem tradicional. A solução implementada leva em consideração alguns parâmetros como: probabilidade de mutação, probabilidade de cruzamento, tamanho da população, dentre outros. A seção 4 possui uma maior detalhamento desses parâmetros, bem como instruções para execução do programa.

3. Implementação

A implementação do algoritmo genético foi realizada na linguagem Java. Como foi utilizada uma linguagem orientada a objetos, vários modelos foram representados em classes como: as rotações da face do cubo, os movimentos possíveis de serem realizados, o cromossomo, dentre outros. A implementação total resultou em 7 pacotes. Para uma melhor representação da implementação, é apresentado a seguir, os diagramas de classes desse algoritmo que estão organizados por pacotes.

3.1. Pacote rotation

O pacote rotation contém as classes responsáveis por realizar as rotações de sentido $90\,^{\circ}$ horário, $90\,^{\circ}$ anti-horário e $180\,^{\circ}$ horário em cada uma das seis faces existentes no cubo.

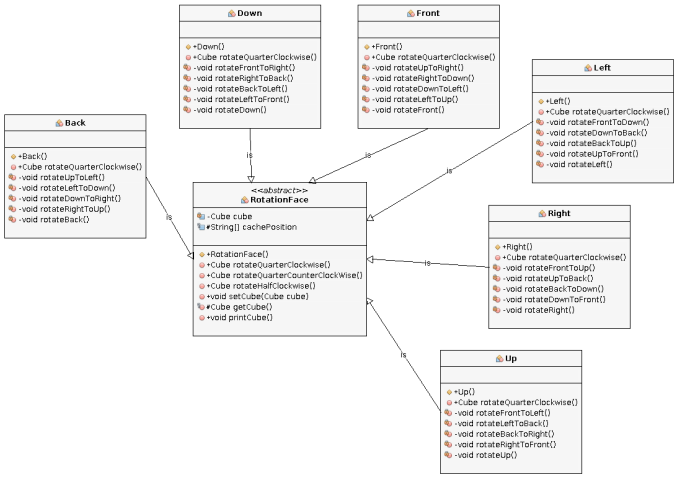

Figura 4. Diagrama de Classes referente ao pacote rotation.

3.2. Pacote structure

O pacote structure contém as classes responsáveis pela estrutura do algoritmo genético. Para um melhor representação, foi criada uma classe para a representação da função Fitness, cromossomo, geração e algoritmo genético. As funcionalidades de cruzamento, mutação, elitismo e torneio, foram modeladas como métodos dentro da classe Genetic Algorithm.

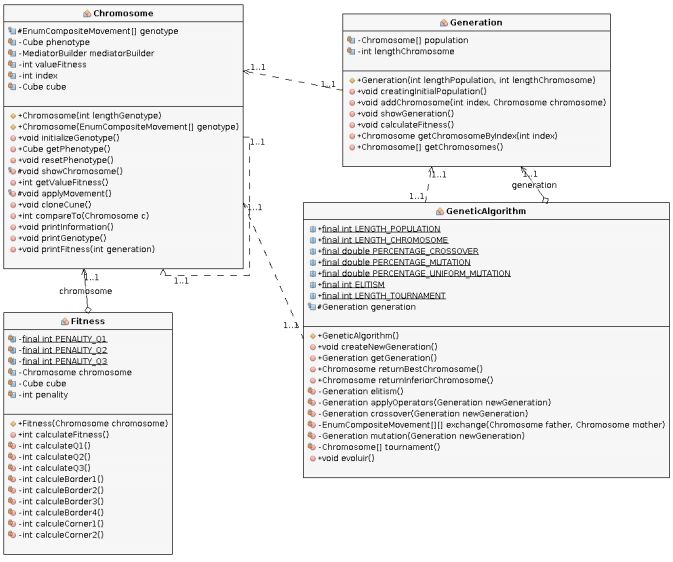

Figura 5. Diagrama de Classes referente ao pacote structure.

3.3. Pacote structure.cube

O pacote structure.cube contém as classes responsáveis pelo cubo. Neste pacote, o cubo é modelado em uma classe, e ainda existe um enum, referente as faces do cubo, para facilitar no gerenciamento do cubo.

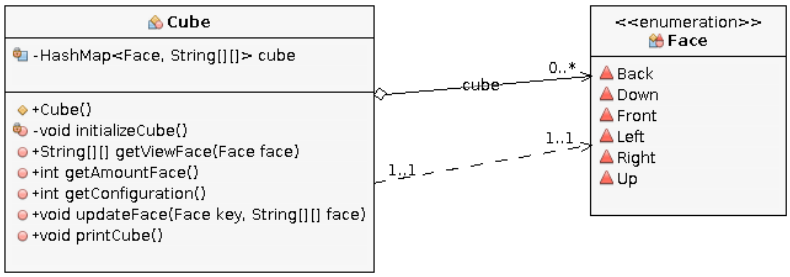

Figura 6. Diagrama de Classes referente ao pacote structure.cube.

3.4. Pacote structure.cube.movements

O pacote structure.cube.movements contém as classes responsáveis por controlar os movimentos do cubo. Para realizar esse controle, foi utilizado o Padrão de Projeto mediator e builder. Esse pacote contém a classe referente ao padrão de projeto Builder (Mediator-Builder), e ainda possui as classes referentes ao padrão de projeto mediator (Mediator, MovementMediator e MovementColleague).

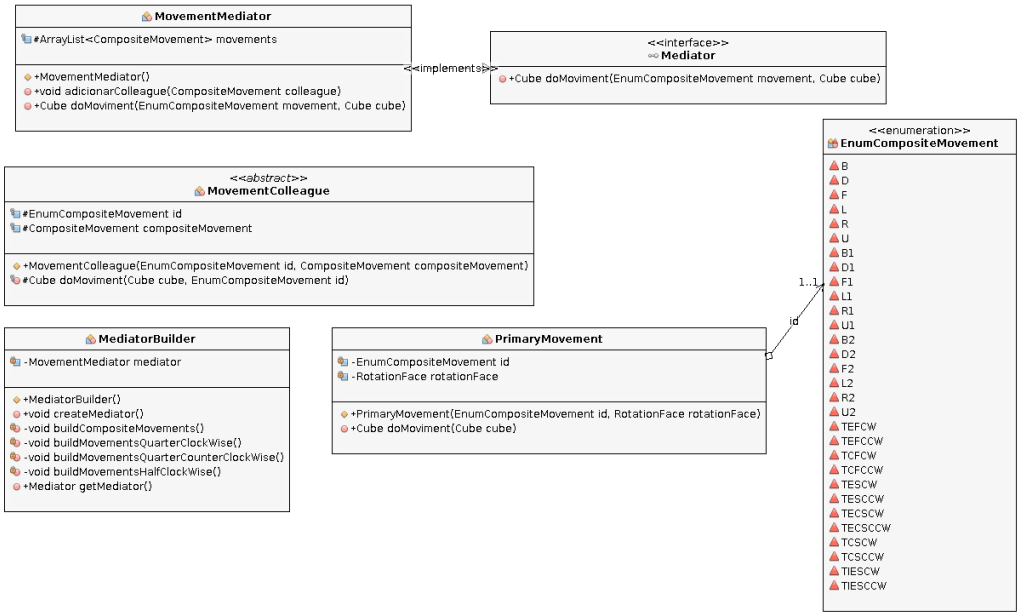

Figura 7. Diagrama de Classes referente ao pacote structure.cube.movements.

3.5. Pacote structure.cube.movements.composite

O pacote structure.cube.movements.composite contém as classes responsáveis por controlar os movimentos do cubo.

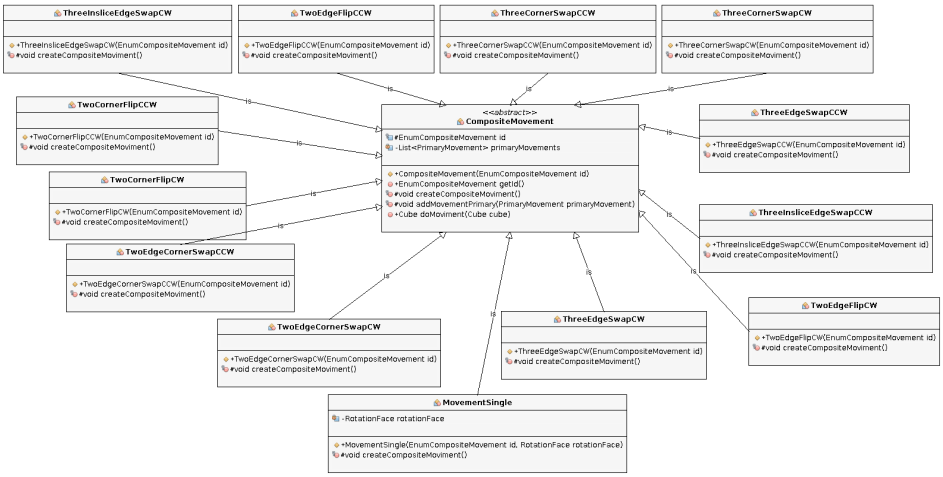

Figura 8. Diagrama de classes do pacote structure.cube.movements.composite.

4. Instruções para Execução do Algoritmo Implementado

O programa foi implementado na linguagem Java e possui todas as classes e pacotes mostradas na seção anterior, dentro da pasta "src". Essa pasta, pode ser facilmente encontrada, uma vez que a mesma está dentro da pasta do projeto. Além das classes que compõe o projeto, existem diversos outros arquivos adicionais nas demais pastas que podem ser interessantes ser visualizados e podem ajudar no entendimento das regras de negócio e funcionalidades deste programa. Dentre esses arquivos adicionais encontram: imagens do diagrama de classes, pequenas documentações, arquivos e gráficos resultantes dos experimentos (seção 5), dentre outros.

Para executar o programa não é necessário realizar nenhum tipo de compilação, devido ao fato do projeto possuir uma versão já compilada por meio da IDE Netbeans, cuja qual foi utilizada para a implementação do algoritmo deste trabalho. A versão compilada desse programa é possível ser encontrada dentro da pasta "dist", presente na pasta do projeto. A versão compilada está no formato de um arquivo ".jar".

4.1. Execução

Para realizar a execução do programa, foi criado um arquivo, chamado: executar.sh. Esse arquivo está situado dentro da raiz do projeto e tem a função de iniciar a execução do programa. Assim, para a realização de testes no projeto, é necessário seguir algumas passos:

- Abra o terminal Linux.
- Vá para a pasta raiz do projeto utilizando o comando cd (digite: cd "nome do diretorio" e pressione enter.
- Dentro do diretório, execute o seguinte comando: **./executar.sh "arquivo de entrada" "arquivo de saída"** ou **sh executar.sh "arquivo de entrada" "arquivo de saída"**

Ao utilizar o comando para execução do programa, é necessário certificar-se de que o aquivo de entrada para a execução do algoritmo está realmente no diretório informado. Caso não esteja, o programa não será inicializado.

Outro ponto muito importante e que deve ser observado ao utilizar esse comando de execução é com relação ao formato dos dados que estão sendo enviados como entrada. O algoritmo espera um padrão de entradas para que ele possa atuar em cima e retorna alguma resposta. Caso, os dados não estejam em conformidade com o padrão exigido, o algoritmo não conseguirá atuar em cima da entrada fornecida. O padrão de entrada exigido por esse algoritmo está especificado na próxima seção.

4.2. Dados de entrada

O padrão de entrada para os dados do algoritmo deve está no seguinte formato:

- Na primeira linha do arquivo, existe um inteiro que indica o tamanho do cubo que o algoritmo deve resolver. Como neste trabalho foi implementado um algoritmo genético que resolve apenas o cubo de tamanho 3 x 3, esse valor inicial será sempre 3 para qualquer arquivo de entrada.
- Na linha 2 do arquivo existe uma string que indica o nome da face que os movimentos passados logo em seguida pertencem. Como um cubo possui seis faces, os possíveis nomes para ser utilizados nessa linha são: Front, Left, Right, Back, Up e Down.
- Na linha de número 3 a 5, contém os movimentos que fazem parte da face especificada na linha anterior, linha 2. Para indicar os movimentos, cada linha deve conter três letras separadas por um espaço que referencie a uma determinada cor. Ex: G, B, B. Lembrando que, as únicas letras que podem ser utilizadas para indicar cores são: B(Azul), G (Verde), O(Laranja), R(Red), W(Branco) e Y(Amarelo).
- Para as próximas linhas do arquivo, o processo se repete (nome da face [sem repetição] em uma linha e uma sequência de movimentos separados por espaços nas próximas três) até que não existem mais faces disponíveis para colocar no arquivo.

O diretório “experiments/tests/instances” possui alguns exemplos de possíveis entradas para o problema e que podem melhor exemplicar o que foi falado nessa subseção.

4.3. Dados de saída

O programa retorna três arquivos de saída. Um dos arquivos retornados é um gráfico que indica a convergência dos indivíduos ao longo das gerações em que a entrada foi submetida. Esse gráfico é muito interessante, pois ele permite visualizar o quanto que uma solução evolui durante as gerações do algoritmo.

O segundo arquivo retornado é uma espécie de log. Ele indica a melhor fitness, a pior e a média da melhor com a pior em todas as gerações. Esse arquivo contém as mesmas informações que o gráfico, porém no gráfico é possível ter noção de como a solução evolui com o decorrer das gerações. No log, é possível avaliar em termos quantitativos a evolução em cada geração.

Por fim o último arquivo gerado indica a solução (a sequência de movimentos para tentar resolver o cubo), na qual o algoritmo genético chegou, e ainda indica a configuração final do cubo após a aplicação final dos movimentos.

5. Experimentos

Esta seção apresenta os planejamentos dos experimentos e testes realizados no algoritmo genético. Os testes foram executados em um computador com processador 2.4 GHz Intel Core i7, com 8GB de memória DDR3 e sistema operacional Xubuntu 14.0.

5.1. Metodologia

Para agilizar a execução dos testes e dos experimentos, foi criado um módulo no próprio programa que realiza todas as execuções de forma automática e emite os relatórios para a análise do melhor valor para um determinado operador. Para cada mudança de um determinado parâmetro, foram realizadas 5 repetições da execução algoritmo, e gerado um gráfico com a média dessas execuções. As avaliações sobre qual valor um determinado parâmetro deve receber foram baseados em cima do gráfico da média.

Várias configurações de parâmetros foram testadas. Porém, neste trabalho é apresentado apenas os gráficos dos valores que apresentaram melhores resultados. Os demais gráficos estão localizado em uma pasta chamada “experiments”, dentro do projeto que possui código fonte da implementação.

5.2. Planejamento dos Testes

Antes de iniciar os testes no algoritmo genético para calibragem de parâmetros, foi realizado um planejamento que consistiu em cinco etapas: variação do tamanho da população e geração, variação dos operadores de cruzamento e mutação, variação do parâmetro de mutação uniforme, variação do tamanho do torneio, e por fim, variação do elitismo.

Cada etapa deste planejamento influencia na etapa seguinte, uma vez que o melhor valor para um determinado parâmetro é fixado ao mesmo na próxima etapa. A seguir é apresentado uma descrição sobre a responsabilidade de cada uma das etapas desse planejamento.

5.2.1. Variação do tamanho da população e geração

Os primeiros experimentos realizados foram com relação ao tamanho da população e geração que melhor se encaixa nesse algoritmo. Apesar de ambos parâmetros estarem descritos juntos em uma mesma subseção, a variação de teste para cada um dos dois foi realizada separadamente. Considerou-se um intervalo de 500 <= p <= 2000 para o tamanho de população e um intervalo de 500 <= g <= 1000 para o tamanho de geração. Como os testes foram realizados de forma automáticas, ambos parâmetros foram intercalados, e, ora altera-se o parâmetro de população, ora alterava-se o parâmetro de geração.

Como os demais parâmetros pertencentes ao algoritmo ainda não foram testes, para realização desse primeiro teste, ambos foram fixados com valores aleatórios, e nas próximas etapas, todos foram reajustados, como ocorrido com os atuais parâmetros em teste.

A **Tabela 3**, exibe a configuração dos parâmetros para o Experimento 1.

Tabela 3. Configuração dos parâmetros para o Experimento 1.

Parâmetros	Configuração
Quantidade de Gerações	500 <= g <= 2000
Tamanho da População	500 <= p <= 1000
Tamanho do Genótipo	20
Taxa de Cruzamento	0,90
Taxa de Mutação	0,15
Taxa de Mutação Uniforme	0,15
Tamanho do Torneio	2
Elitismo	1

5.2.2. Variação dos operadores de cruzamento e mutação

Esta etapa do experimento tem por objetivo avaliar qual a melhor configuração para a taxa de cruzamento e mutação. Nessa parte do experimento, a melhor configuração para os parâmetros de população e geração são fixados com base no Experimento 1. Apesar de cruzamento estarem sendo avaliados nessa mesma seção, a variação de teste para cada um dos dois foi realizada separadamente. Considerou-se um intervalo de 0.1 <= c <= 0.9 para a taxa de cruzamento e um intervalo de 0.01 <= m <= 0.51 para a taxa de mutação.

A **Tabela 4**, exibe a configuração dos parâmetros para o experimento 2.

Tabela 4. Configuração dos parâmetros para o Experimento 2.

Parâmetros	Configuração
Quantidade de Gerações	1500
Tamanho da População	1000
Tamanho do Genótipo	20
Taxa de Cruzamento	0,10 <= c <= 0,90
Taxa de Mutação	0,01 <= m <= 0,50
Taxa de Mutação Uniforme	0,15
Tamanho do Torneio	2
Elitismo	1

5.2.3. Variação do parâmetro de mutação uniforme

Esta etapa do experimento tem por objetivo avaliar qual a melhor configuração para a taxa de mutação uniforme. Nessa parte do experimento, a melhor configuração para os parâmetros de cruzamento e mutação são fixados com base no Experimento 2, bem como os parâmetros de população e geração já fixados no experimento anterior. Considerou-se um intervalo de 0.1 <= mu <= 0.5 para a taxa de mutação uniforme.

A **Tabela 5**, exibe a configuração dos parâmetros para o experimento 3.

Tabela 5. Configuração do parâmetro para o Experimento 3.

Parâmetros	Configuração
Quantidade de Gerações	1500
Tamanho da População	1000
Tamanho do Genótipo	20
Taxa de Cruzamento	0,30
Taxa de Mutação	0,15
Taxa de Mutação Uniforme	0,10 <= mu <= 0,50
Tamanho do Torneio	2
Elitismo	1

5.2.4. Variação do tamanho do torneio

Esta etapa do experimento tem por objetivo avaliar qual a melhor configuração para o tamanho do torneio. Nessa parte do experimento, a melhor configuração para os parâmetros de mutação uniforme, cruzamento, mutação, população e geração estão fixados. Considerou-se um intervalo de 2 <= t <= 10 para a taxa de mutação uniforme.

A **Tabela 6**, exibe a configuração dos parâmetros para o experimento 4.

Tabela 6. Configuração dos parâmetros para o Experimento 4.

Parâmetros	Configuração
Quantidade de Gerações	1500
Tamanho da População	1000
Tamanho do Genótipo	20
Taxa de Cruzamento	0,30
Taxa de Mutação	0,15
Taxa de Mutação Uniforme	0,15
Tamanho do Torneio	2 <= t <= 10
Elitismo	1

5.2.5. Variação do elitismo

Esta etapa do experimento tem por objetivo avaliar se o fator elitismo tem influência na qualidade da solução final. Nessa parte do experimento, a apenas o parâmetro elitismo é variado. Os demais parâmetros já foram todos testados nos experimentos anteriores, e com isso permanecem fixos nessa altura dos testes. Considerou-se um intervalo de 0 <= e <= 1 para o parâmetro de elistismo.

A **Tabela 7**, exibe a configuração dos parâmetros para o experimento 5.

6. Resultados

Esta subseção tem como objetivo mostrar os resultados obtidos após cada execução dos experimentos destacados na subseção anterior.

Tabela 7. Configuração dos parâmetros para o Experimento 5.

Parâmetros	Configuração
Quantidade de Gerações	1500
Tamanho da População	1000
Tamanho do Genótipo	20
Taxa de Cruzamento	0,30
Taxa de Mutação	0,15
Taxa de Mutação Uniforme	0,15
Tamanho do Torneio	3
Elitismo	0 <= e <= 1

6.1. Resultado do Experimento 1

Após executar as repetições de testes para esse experimento, foi possível encontrar um valor para a quantidade de geração e quantidade de população bons para o algoritmo.

A **Figura 9** mostra um gráfico com os resultados do melhor indivíduo obtido para cada a partir da média das repetições. As cores presentes no gráfico representam as gerações, que possui dois valores diferentes utilizados no teste: 500 e 1000.

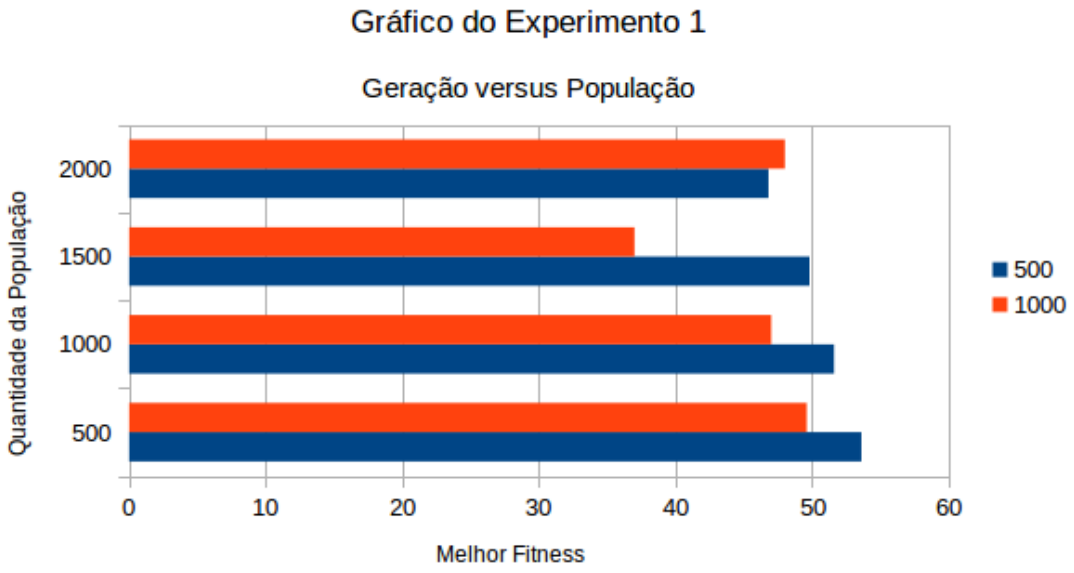

Figura 9. População x Geração.

Ao analisar o gráfico percebe-se que o tamanho da população no algoritmo genético tem uma grande influência na qualidade da solução. Quando foi realizado testes com tamanho pequeno da população (500 e 1000), os indivíduos na apresentaram bons resultados. Principalmente quando utilizou-se apenas 500 gerações para os indivíduos evoluírem. Isso mostra que a quantidade de geração é um outro fator que afeta na qualidade da solução.

Resolver o tentar resolver o cubo de Rubik é algo bastante complexo devido a grande quantidade de movimentos que é possível realizar em um cubo, e ao grande espaço

de busca existente nesse problema. Quando um problema possui uma alta complexidade, o tamanho da população e a quantidade de gerações tem grande influência no resultado da solução, uma que os indivíduos não possuem um tempo suficiente para evoluírem e acabam convergindo rapidamente. Pelo gráfico é possível perceber um melhor comportamento das soluções, a partir do momento em que esta assume um tamanho de 1500 indivíduos.

Apesar de ter grande importância em problemas de grande complexidade, a população e as gerações não podem ser exageradamente altas. Valores muito altos, além de prejudicar na convergência da solução, também prejudica no tempo de execução do algoritmo, deixando-o mais lento. No gráfico acima, é possível perceber que a configuração que se destacou nesse experimento foi: tamanho da população igual a 1500 e número de gerações igual a 1000. Esses parâmetros são fixados no próximo experimento.

6.2. Resultados do Experimento 2

O segundo experimento ajuda a buscar melhores valores para os operadores de cruzamento e mutação. Após a execução deste experimento, ficou claro que o aumento da mutação ajudar a introduzir diversidade nas populações de cada geração. Como existem muitas combinações de movimentos, uma taxa alta de cruzamento pode fazer com que a população tenha uma convergência rápida e encontre um ótimo local.

Para evitar esse acontecimento e tentar trazer mais diversidade à população, a taxa de mutação precisa ser um pouco mais alta e o cruzamento possuir uma taxa pequena.

A **Figura 10** mostra com os resultados do melhor indivíduo obtido através da média das repetições. As cores presentes no gráfico representam indicam as seguintes taxas: 0,01; 0,1; 0,15; 0,3 e 0, 5.

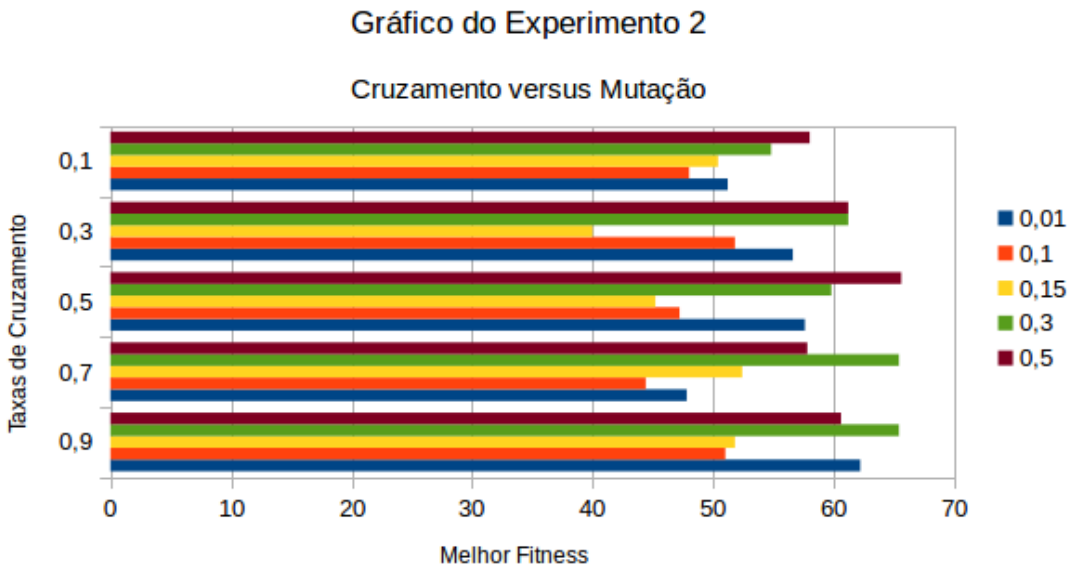

Figura 10. Cruzamento x Mutação.

Ao analisar o gráfico, percebe-se que a taxa de mutação com uma probabilidade de 15% e o cruzamento com uma probabilidade baixo, 30% prevaleceram com bons re-

sultados. O principal motivo que pode explicar a soberania dessa configuração nos testes é a relação com diversidade. Uma taxa de cromossomos muito alta tende a igualar os indivíduos muito rápido, evitando assim diversidade na população. A taxa de cruzamento mais reduzido nesse problema, faz com que os indivíduos não sejam igualados muito rápido, aumento assim a chance de encontrar soluções interessantes na população.

6.3. Resultado do Experimento 3

O terceiro experimento ajuda a buscar melhores valores para o operador de mutação uniforme, o qual tem como objetivo verificar a probabilidade de realizar a mutação em cada gene de um cromossomo. Após a execução deste experimento, percebeu-se que esse operador não é necessário conter uma taxa muito alta de probabilidade de mutação.

A **Figura 11** mostra os resultados do melhor indivíduo obtido através da média das repetições. O experimento utilizou cinco tipos diferentes de taxas de mutação: 0,01; 0,1; 0,15; 0,30 e 0,50.

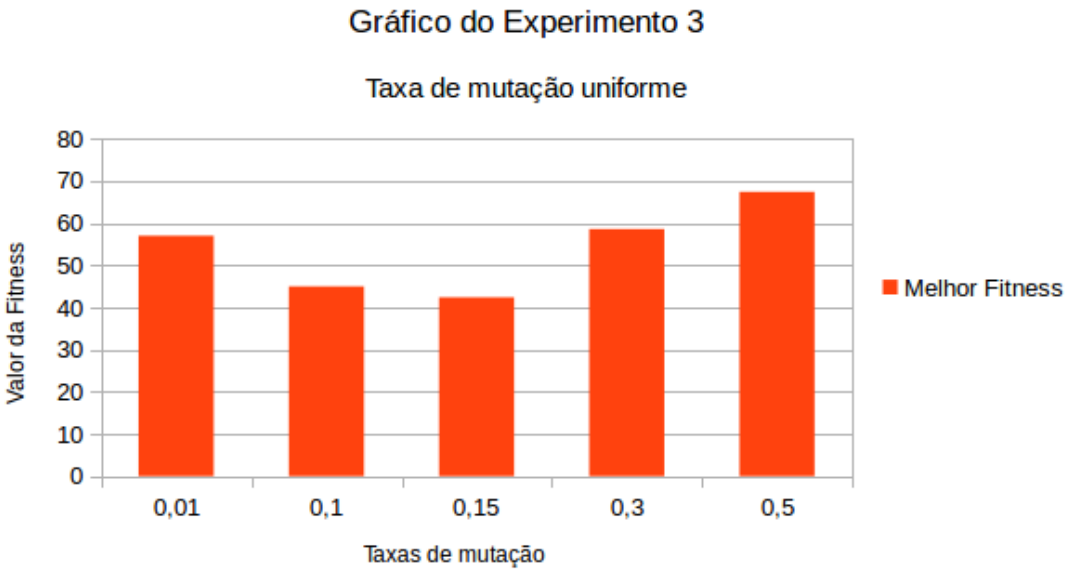

Figura 11. Taxa de mutação uniforme.

Ao analisar o gráfico, percebe-se que a taxa de mutação com uma probabilidade de 15% produz boas soluções.

6.4. Resultado do Experimento 4

No quarto experimento foram testados diferentes tipos de torneios e no final, avaliou-se o torneio que apresentou melhor resultado. Para esse experimento foram utilizados os seguintes tamanhos de torneio: 2, 3, 5, 7 e 10.

A **Figura 12** mostra o resultado da fitness, obtidas dessas seguintes variações de tamanho do torneio.

Como é possível observar no gráfico, o torneio de tamanho três foi o que apresentou um melhor resultado nas simulações realizados, e por isso ele foi fixado aos parâmetros do algoritmo genético deste trabalho.

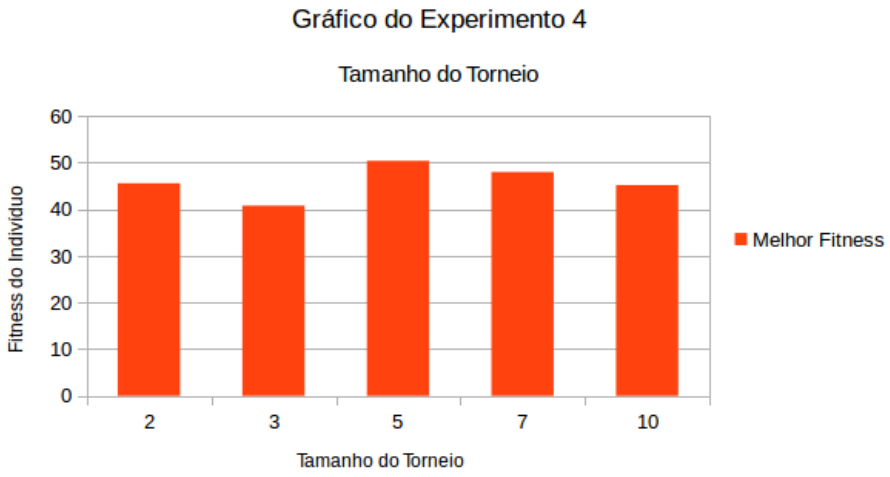

Figura 12. Tamanho do torneio.

6.5. Resultado do Experimento 5

No quinto experimento foi realizado testes com elitismo, a fim de saber se esse operador poderia ou não influenciar na solução. Ao finalizar os testes, percebeu-se que o elitismo possui grande influência na qualidade da solução e não pode ficar de fora da solução. Esse operador garante que a solução nunca fique ruim. No pior cenário, a solução sempre se manterá constante.

A **Figura 13** mostra o gráfico com os resultados deste experimento.

Figura 13. Experimento com elitismo.

6.6. Testes

Após a realização dos experimentos e configuração dos parâmetros, o algoritmo genético foi testado para três instâncias diferentes: in1, in2 e in3. Nos gráficos abaixo, a linha azul representa a solução encontrada pelo algoritmo em cada uma das gerações e a linha vermelha representa a solução ótima do problema:

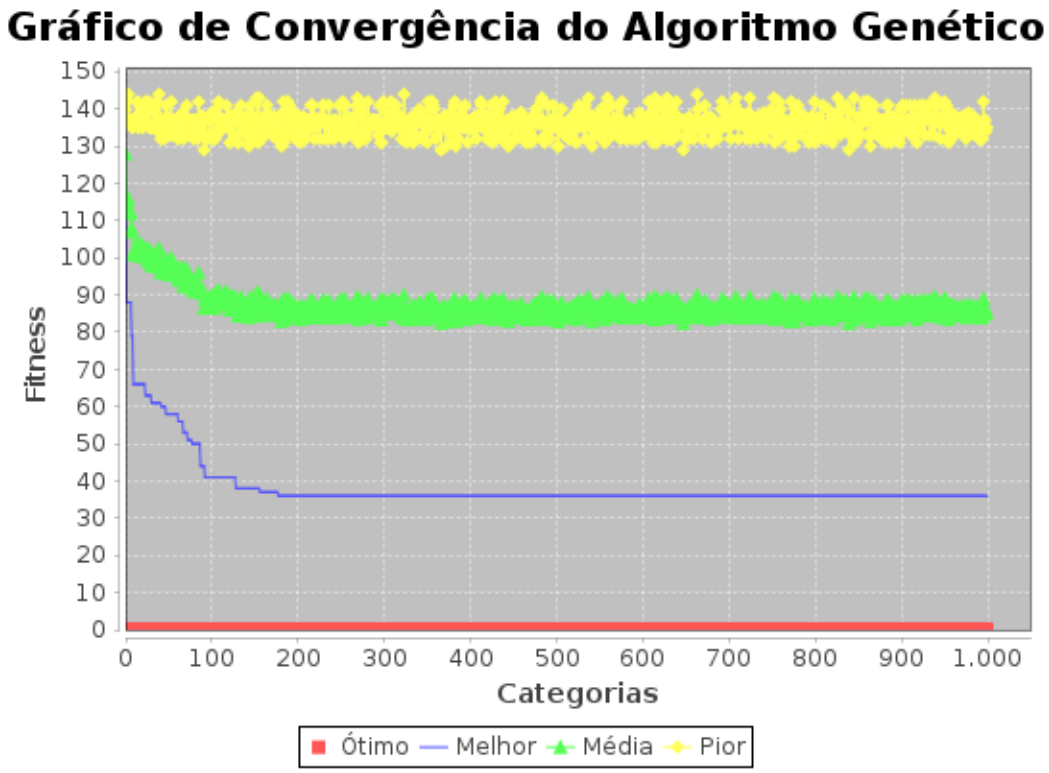

Figura 14. Teste realizado para a instância in1.

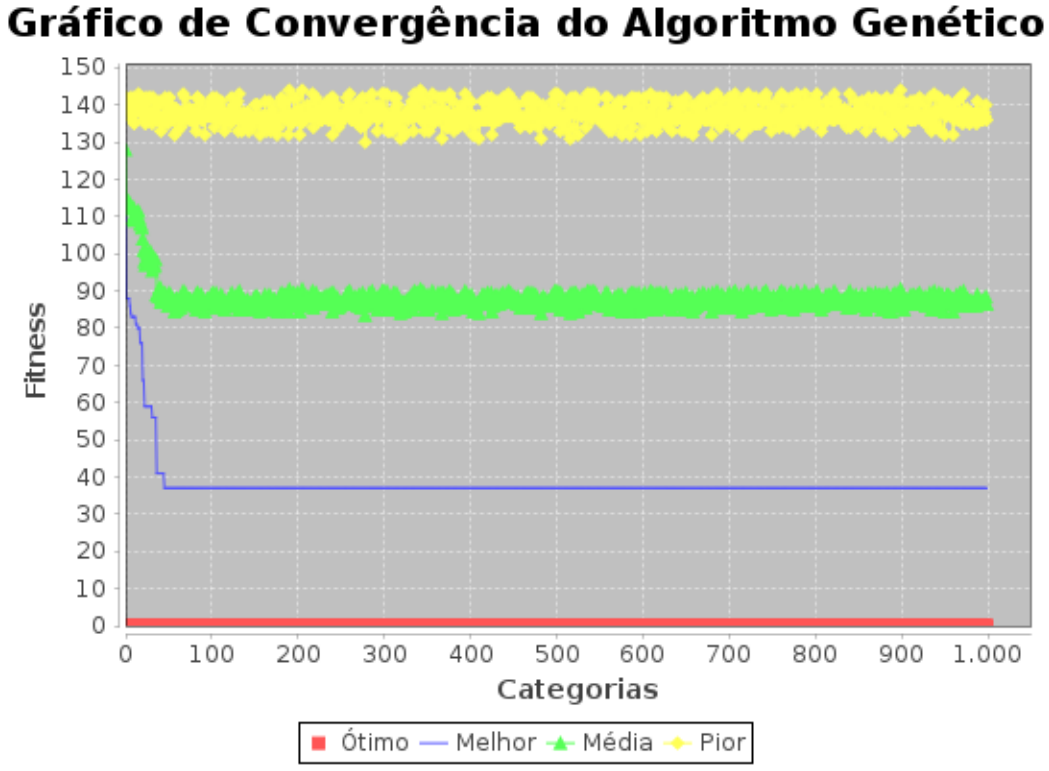

Figura 15. Teste realizado para a instância in2.

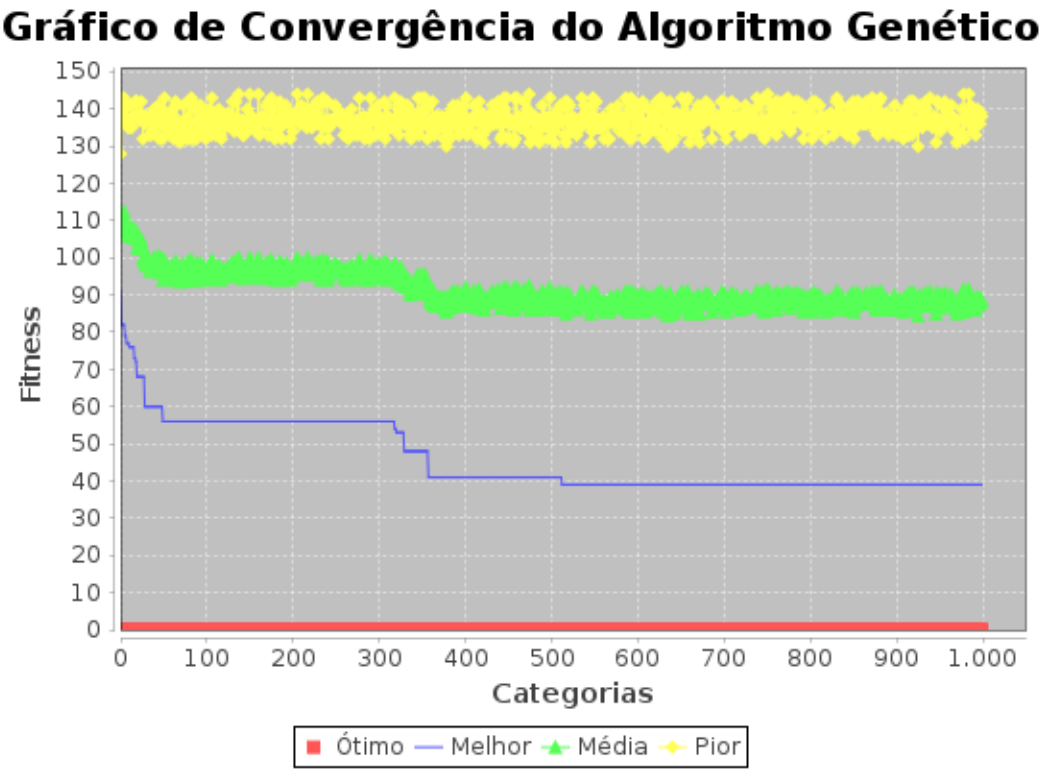

Figura 16. Teste realizado para a instância in3.

7. Conclusão

Resolver o cubo mágico não é uma tarefa tão trivial. Seu espaço de busca exponencial é um dos principais desafios na resolução desse jogo. Devido ao fato desse jogo tratar-se de uma análise combinatória, uma solução baseada em um algoritmo genética talvez seja uma boa tentativa para resolução desse jogo, uma vez que esses algoritmos são destinados a problemas de otimização.

Baseado nisso, este trabalho voltou-se para a aplicação de um algoritmo genético para resolver o cubo mágico. Para atingir o objetivo, foi necessário realizar a modelagem e ainda realizar diversos experimentos para validação e tunning dos parâmetros e operadores deste algoritmo, uma vez que, essa estratégia é fundamental para o sucesso desse algoritmo.

Porém, como foi falado no início dessa seção, resolver o cubo mágico não é uma tarefa fácil, mesmo quando aplica-se uma estratégia computacional. Embora nos testes realizados não foi possível chegar a resolução deste jogo, o algoritmo mostrou um bom desempenho e conseguiu evoluir essas entradas, chegando perto de um possível resolução do cubo.

Portanto, algoritmos genéticos podem ser uma boa estratégia para a resolução do cubo mágico, porém, é necessário que sejam haja novos tipos de operadores de cruzamento e mutação para ajudar esse algoritmo no processo de evolução deste jogo.

Referências

Herdy, M. and Patone, G. (1994). Evolution strategy in action, 10 es-demonstrations. In Spring, editor, *Parallel Problem Solving from Nature, PPSN XI*, pages 442–451. 11th International Conference Krákow.