Algoritmo Genético

Sávio Lage, João Bastos

saviohenrique.lage@unifei.edu.br
jvpereirabastos@unifei.edu.br

Palavras Chave: Algoritmo, Genético, Labirinto, População, Individuo

Abstract

This is where the abstract should be placed. It should consist of one paragraph and a concise summary of the material discussed in the article below. It is preferable not to use footnotes in the abstract or the title. The acknowledgement for funding organisations etc. is placed in a separate section at the end of the text. We wish you success with the preparation of your manuscript.

Resumo

No presente artigo tratamos da aplicação de um algoritmo genético para solucionar labirintos, gerando assim um caminho resposta. Proposto pelo professor Sandro Carvalho durante a disciplina de Inteligência Artificial da Universidade Federal de Itajubá - *Campus* Itabira, temos na entrada do algoritmo, um labirinto formado pelos caracteres "+", -", "e salvo em um documento ".txt". Para saída deveríamos apresentar uma solução de tal labirinto representada por um indivíduo. Para isso criamos uma lógica de conversão de um labirinto em ".txt"para uma matriz de zeros e uns. Após isso geram-se os indivíduos, que são representados por um vetor, também de zeros e uns, que cresce a medida que o indivíduo percorre os caminhos possíveis do labirinto. Por conseguinte aplicam-se os operadores genéticos, característicos de um GA, aos indivíduos, para convergirmos para solução.

1 Introdução

Inspirado na teoria da evolução, do cientista Charles Darwin,[**?**] um algoritmo genético é uma metaheurística frequentemente utilizada para gerar soluções de alta qualidade em problemas de busca e otimização. Introduzido por Jhon Holland[3] , o algoritmo reflete o processo de seleção natural que utiliza operadores genéticos conhecidos da biologia, tais como: mutação, cruzamento e seleção, em que o indivíduo com melhor aptidão - em inglês (*fitness*) - se reproduzirão, gerando indivíduos cada vez melhor.[1]

Podemos explicar algoritmo em cinco principais fases: População Inicial - a heurística começa com uma geração de indivíduos aleatórios, que possuem seus genes incorporador por um vetor de zeros e uns usualmente chamado cromossomos, em que cada cromossomo representa uma possível solução, e o conjunto de cromossomos uma população.

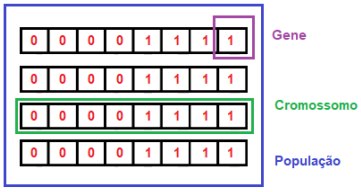

Figura 1. Representação dos indivíduos - Fonte: Autores

Função de aptidão (*fitness*) - ela determina, o quão bem um indivíduo se comporta para resolver o problema apresentado baseado em seus genes, e seu valor é determinante para a fase de seleção. Cabe ao problema determinar qual função seria utilizada, em que não necessariamente a maior (*fitness*) significaria o melhor indivíduo. Cruzamento (*crossover*) - o cruzamento pode ser considerado um intensificador de características, em que geralmente, se baseia em selecionar a primeira metade dos genes de um cromossomo, com a segunda metade de um outro cromossomo (corte simples), gerando novos indivíduos dois a dois.[2]

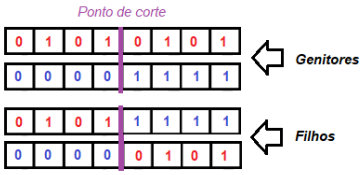

Figura 2. Crossover com corte simples - Fonte: Autores

Mutação - é em grande parte, a responsável pela variedade genética dos indivíduos a cada geração, fun-

damentada em alterar algum gene, de zero para um, ou contrário, modificando a (*fitness*) do indivíduo. Entretanto não se deve aplicar uma alta taxa de mutação, pois isso poderia ocasionar em uma aleatoriedade muito alta nos indivíduos, reduzindo muito a convergência do algoritmo, prejudicando a eficiência do mesmo.[2]

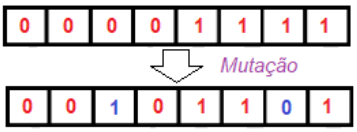

Figura 3. Representação de mutação - Fonte: Autores

Após as operações genéticas realizadas, temos uma nova geração do mesmo tamanho da população inicial, agora, com indivíduos que herdam as boas características dos seus genitores - estes são chamados filhos e majoritariamente possuem aptidões mais elevadas. Esse processo se repete até a a população tenha convergido para a condição de parada, que no problema do labirinto abordado a seguir é o local de chegada.[4]

```
INICIAR
Geração da população inicial
Calcular valores de aptidão
REPETIR
    Seleção
    Cruzamento
    Mutação
    Calcular nova aptidão
ENQUANTO População não convergir
PARAR
```

Figura 4. Pseudocódigo de um Algoritmo Genético - Fonte: Autores

2 Materiais e Métodos

Para desenvolvimento deste projeto foi utilizada a linguagem Pyhton, por possuir uma sintaxe fácil e ter bibliotecas já otimizadas. Neste código, contamos com o uso de bibliotecas como "pyGame"para desenvolver a interface gráfica do labirinto e do caminho percorrido pelo individuo, "copy"para a realização de cópias das listas que são passadas automaticamente como referencias entre as funções, "random"para a geração e escolha de elementos aleatórios ao longo do código, "timeit"para importar um contador de tempo e medir a resolução do labirinto em segundos, "os"e "sys"para controle de eventos de sistema e também "json"para manipulação do arquivo .json gerado contendo os resultados e dados sobre estes.

O objetivo deste trabalho é resolver os labirintos propostos utilizando principios de algoritmos genéticos, para isso, foram necessárias o desenvolvemto de algumas etapas

2.1 Tratamento do arquivo texto

O algoritmo inicia com uma interação com o usuário, perguntando qual o nome do arquivo que este deseja executar. Cada arquivo representa um labirinto feito em arquivo de texto conforme figura 5

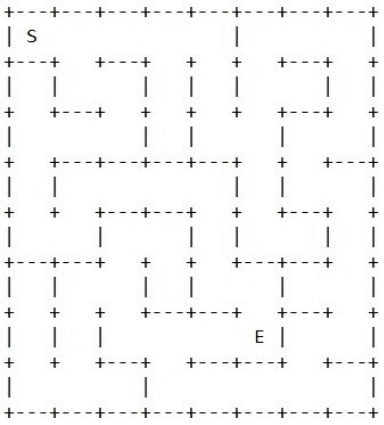

Figura 5. Labirinto em formato texto - Fonte: Sandro

Em seguida, o programa executa as rotinas da classe *HandleFile*, desenvolvida pelos alunos, abrindo o arquivo escolhido e inicia o tratamento do texto, separando cara linha em um elemento de uma lista e cada elemento dessa lista é separado em um caractere, por fim, tem-se uma matriz de caracteres. Iterando sobre esta matriz, converte-se os caracteres em números conforme a tabela 1

Valor	Descrição
0	Espaço disponível
1	Parede
2	Ponto Inicial
3	Ponto Final

Tabela 1. Tabela de valores da matriz labirinto - Fonte: Autores

Por fim, elimina os elementos de colunas ímpares, visto que estes são apenas espaços duplicados aplicados para tornar mais simples a visualização do labirinto em arquivo texto. Por fim, convertendo os números em pixels coloridos em uma janela, temos um resultado conforme a figura 6, sendo a área cinza como parede, área em preto como caminho disponível, ponto vermelho representando o ínicio e azul o final. Nos resultados, é mostrada esta mesma janela contando um

caminho em amarelo, representando a rota até a solução.

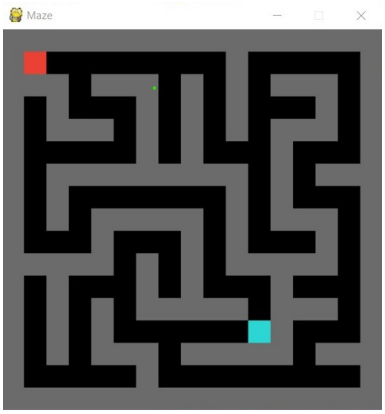

Figura 6. Labirinto em formato gráfico - Fonte: Autores

2.2 População e Cromossomo

Como elementos básicos em um algoritmo genético, é necessário a presença de elementos como individuos (cromossomos) e população. Neste algoritmo, o individuo é formado por uma classe contendo uma variavel "road"que contém as direções que este irá seguir, conforme a tabela 2

Valor	Descrição
1	Esquerda
2	Cima
3	Direita
4	Baixo

Tabela 2. Tabela de valores da matriz labirinto - Fonte: Autores

Além disso, contem uma variável "rating"que vai armazenar a nota do inidividuo, sendo esta gerada pela função fitness, função que mede sua aptidão no labirinto. A variável "possibleRoads"armazena as possiveis direções que o individuo pode percorrer em cada posição. A variável "indexesOfPossibleRoads"armazena a lista de indices da lista "possibleRoads"em que o individuo tem mais opções de caminho. A variável "generation"armazena a geração em que o individuo se encontra, "hadACollision"é uma variável booleana que verifica se aquele individuo colidiu com a parede, ou não, e a variável "found"verifica se o individuo encontrou a solução ou não.

Já a classe população possui apenas duas variáveis, sendo elas, "individuals"contendo uma lista de individuos e "size", contendo seu tamanho. Vale ressaltar que esta é gerada de forma dinamica, onde o número de individuos é definido pelo usuário ao inicio do programa.

2.3 Execução

Como já explicado, o programa inicia requisitando o usuário o nome do arquivo que contém a matriz, após isso, a interação com o usuário continua para que este defina a quantidade máxima de gerações que o algoritmo executará e o tamanho da população presente. Após isso, é iniciado um processo de busca pela solução. A cada geração ocorre uma iteração sobre todos os indivíduos da população, sua geração é atualizada, gera-se as possíveis direções que ele pode percorrer e o move para este novo ponto, em caso de mais de uma opção a escolha é feita de forma aleatória, sua nota é calculada pela função fitness, que é basicamente um contador de espaços já percorridos pelo individuo e sua mutação ocorre baseada em uma porcentagem, este valor representa a chance de um individuo modificar um de seus últimos pontos de cruzamento, e todos os outros pontos subsequentes a este.

Vale ressaltar que o individuo só caminha uma unidade da matriz por geração. A figura 7 especifica um individuo na matriz da figura 6 após quatro gerações. Partindo do ponto inicial, sua única opção disponível é caminhar pela direita, valor 3, e esta é executada. Nas duas próximas gerações, esta operação é repetida. Quando inicia a quarta geração o individuo encontra-se em um ponto denominado cruzamento, onde ele tem duas opções de caminho, caminhar para a direita ou para baixo, com isso, adiciona-se este índice a lista que armazena os índices dos pontos de cruzamento.

Figura 7. Primeiras gerações - Fonte: Autores

2.4 Gerações

Para este desenvolvimento, foi definido uma série de funções para a resolução do problema, primeiramente atualiza-se a geração, em seguida itera sobre cada individuo da população, define sua geração atual, gera uma lista de possíveis pontos que este pode percorrer, baseado em sua localização atual, anda com o individuo para esta posição, em caso de mais de uma opção a escolha é feita de forma aleatória, calcula sua nota baseado na quantidade de passos do individuo e por fim, realiza a mutação com uma porcentagem de ocorrência. Após a iteração sobre todos os indivíduos, verifica-se se a resolução foi encontrada e armazena o melhor individuo já gerado. O pseudo-código, em python, para a execução em gerações pode ser exemplificado da seguinte forma:

```
for currentGeneration in range(generations):
    for individual in population.individuals:
        individual.setGeneration(currentGeneration)
        individual.generateIndexesArrays(initialCoordinates, maze)
        individual.appendRoad(individual.possibleRoads[-1])
        individual.fitness(initialCoordinates, maze)
        individual.mutation(mutationChance)
        if individual.found:
            bestIndividual.saveIndividual(individual)
            break
    if bestIndividual.found:
        break
```

2.5 Mutação e Cruzamento

Mutação e cruzamento são termos recorrentes quando estuda-se algoritmos genéticos. Essa modificação dos elementos tem como principal objetivo diversificar a população e fazer com que está possua melhor aptidão com o passar das gerações. Neste trabalho a função mutação consiste em uma heurística de *backtracking* aplicada nos pontos onde o individuo pode assumir múltiplos caminhos, eliminando assim os caminhos que direcionam a um erro já cometido por estes. Vale ressaltar que esta lógica é aplicada mediante um valor percentual para seu acontecimento e que não há *feedback* entre os indivíduos. Cada um possui suas próprias características e caminhos independentes.

Embora presente nas premissas de um algoritmo genético, o cruzamento não é utilizado para a resolução deste problema. Após estudos e testes, verificou-se que não há necessidade do uso desta técnica, visto que, os indivíduos se movimentam somente pela área disponível e um processo de cruzamento poderia alterar esta situação, quando selecionados dois indivíduos de caminho com pontos iniciais distintos.

3 Resultados

O programa Foi capaz de gerar resultados para três das quatro matrizes propostas. Arquivo de nome "M0"contem uma matriz quadrada de tamanho 17, sua resolução compreende um total de 34 passos com tempo médio de 1,2 segundos de conclusão

O programa permite que o usuário execute N vezes o algoritmo, sendo N um número definido ao inicio da execução. Após completar todas as N resoluções, é gerado um arquivo contendo os resultados e seus dados, como por exemplo a nota do melhor individuo daquela execução, se este encontrou ou não o labirinto e o tempo, em segundos de cada execução. Um exemplo de labirinto resolvido é retratado na figura 8, onde tem-se o labirinto presente no arquivo 'M1.txt' representado de forma gráfica e o caminho em amarelo é descrito como o caminho realizado pelo individuo que solucionou o problema.

Além da janela de resolução, é gerado um arquivo em formato json contendo informações a respeito da execução do código e do melhor individuo encontrado nesta, como por exemplo, sua nota gerada pela função fitness, em qual geração que esta resolução se encontra, se o final do labirinto foi descoberto, o tempo de execução do código e a quantidade de pontos de múltiplas opções de caminho da resposta. Vale ressaltar que a nota do individuo representa o tamanho de sua rota.

Após estudos e testes de performance, ficou definido um valor fixo de 10.000 gerações como quantidade máxima e uma população de 25 indivíduos. Assim, todas as resoluções demonstradas contam com estes parâmetros.

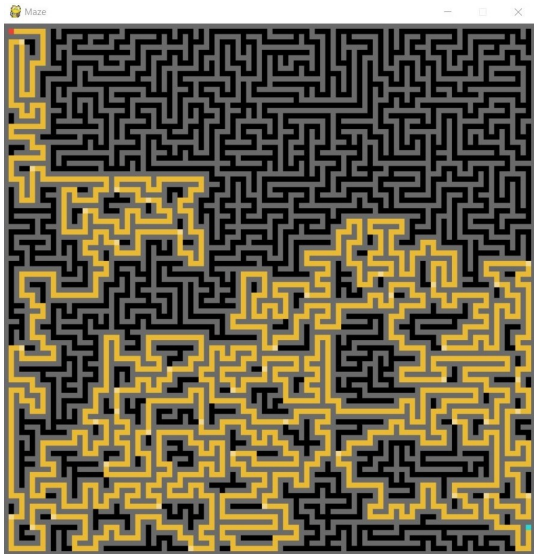

Figura 8. Resolução para 'M1'- Fonte: Autores

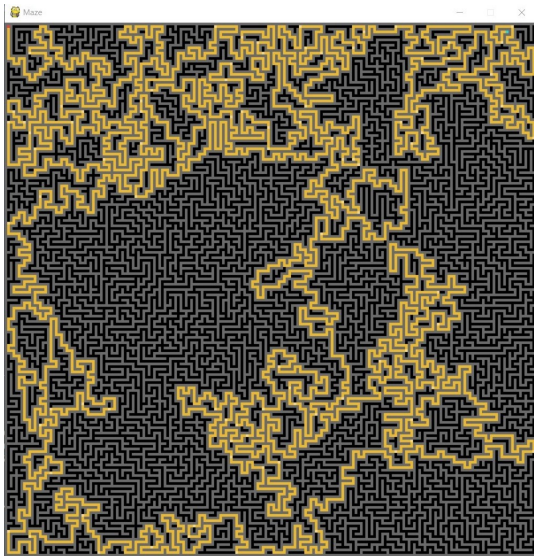

Figura 9. Resolução para 'M2'- Fonte: Autores

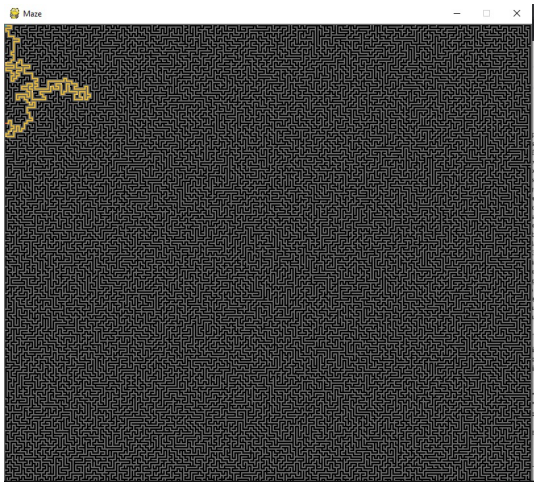

Figura 10. Resolução para ’M3’- Fonte: Autores

A partir da análise dos dados das três matrizes, obteve-se os seguintes indicadores:

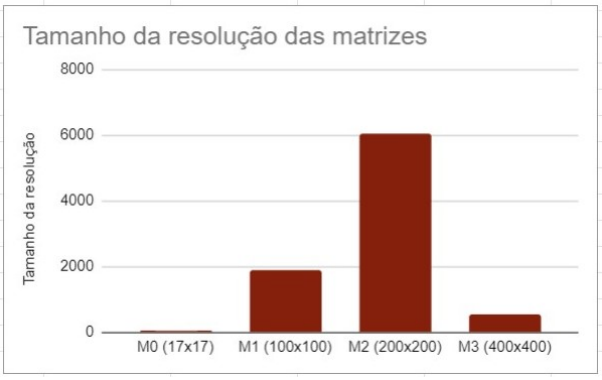

Figura 11. Comparativo tamanho das resoluções- Fonte: Autores

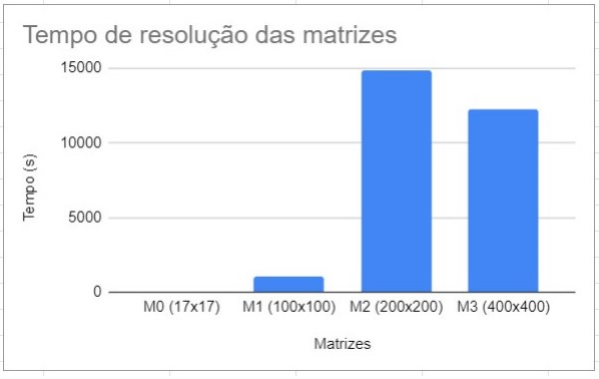

Figura 12. Comparativo de tempo das resoluções - Fonte: Autores

Vale ressaltar a alteração na tendencia crescente dos gráficos, analisando respectivamente M0, M1 e M2, nota-se uma relação proporcional exponencial entre tamanho da matriz, tempo de resolução e tamanho de resolução. Já a matriz M3 possui um tamanho de resposta inferior à M1, contudo com um tempo próximo à resolução da M2, mesmo apresentando o dobro do tamanho, em largura.

4 Considerações Finais

Durante o desenvolvimento do algoritmo genético para o problema do labirinto, encontramos diversos desafios para que os indivíduos convergissem de forma eficaz para o ponto objetivo. As particularidades de maior destaque para o GA apresentado, foram as premissas de que os indivíduos apenas seguiam caminhos possíveis, reduzindo assim a quantidade de indivíduos que encontravam paredes seguidamente, prejudicando a convergência. Além da proposição de armazenar os pontos de encruzilhada dos indivíduos, que permite que os mesmos não fiquem presos a um beco sem saída. Conclui-se portanto, que ao adotarmos variadas técnicas de otimização para o algoritmo genético, é possível obter resultados satisfatórios utilizando esta metaheurística com o intuito de encontrar uma solução para um labirinto.

Referências

[1] Vijini Mallawaarachchi. Introduction to Genetic Algorithms . In *Towards Data Science*, Australy, 2017.

[2] Melanie Mitchell. Genetic Algorithms: An Overview. *Cambrige*, 1995.

[3] Melanie Mitchell. An Introduction to Genetic Algorithms. *Cambrige*, 1996.

[4] Mathworks Team. What Is the Genetic Algorithm? . In *Towards Data Science*, https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html", 2017.