Computação Evolutiva na resolução de Labirintos

Bruno M. Pires[1]

[1]Departamento de Ciências da Computação
Universidade do Estado de Santa Catarina (UDESC)
Caixa Postal 15.064 – 88.035-901 – Joinville – SC – Brasil

bruno.pires@edu.udesc.br

Abstract. *This report presents an effective solution to a classic problem of finding the exit in a maze. The approach uses genetic algorithms to find optimal paths for a robot in an NxM maze, avoiding collisions. Promising results indicate the viability of this technique in solving this optimization challenge.*

Resumo. *Este relatório apresenta uma solução eficaz para um problema clássico de encontrar a saída em um labirinto. A abordagem utiliza algoritmos genéticos para encontrar caminhos ideais de um robô em um Labirinto NxM, evitando colisões. Resultados promissores indicam a viabilidade dessa técnica em resolver esse desafio de otimização.*

1. Introdução ao problema do labirinto

Labirintos são estruturas intrigantes e responsáveis por originar diversas obras, desde jogos até mesmo filmes e livros. No entanto, na Ciência da Computação, os labirintos assumem um papel muito mais pragmático e desafiador. Eles representam um campo de estudo que envolve a resolução de problemas complexos relacionados à navegação, otimização, busca e algoritmos.Os labirintos, que podem ser tanto físicos quanto virtuais, oferecem uma rica variedade de desafios computacionais que vão desde encontrar o caminho mais curto entre dois pontos até explorar ambientes desconhecidos de maneira eficiente.

Seguindo esta linha, este relatório será responsável por aplicar um algorimto evolutivo na resolução de alguns labirintos que serão apresentados a seguir. O objetivo principal do algoritmo é encontrar a saída com menor caminho possível, sendo este o chamado ”resultado ótimo”.

O Relatório está estruturado da seguinte forma, uma seção dedicada a apresentar as características de desenvolvimento da ferramenta, uma seção apresentando os resultados dos casos de teste, e uma seção de conclusão.

2. Desenvolvimento

Assim como no desenvolvimento de software, inicialmente foi necessário realizar uma espécie de ”modelagem” do problema, pra enteder melhor suas características. De forma básica, a tabela mostrada abaixo foi preenchida, utilizando os conceitos também mostrados abaixo:

- Agentes: O agente é a entidade que toma decisões e interage com o ambiente.

- Observável: Refere-se à capacidade do agente de observar o estado atual do ambiente.
- Determinístico: Indica se o comportamento do agente é completamente previsível com base no estado atual e nas ações tomadas.
- Episódico: Indica se o agente toma decisões em episódios ou etapas discretas, onde as ações tomadas em um episódio não afetam diretamente os episódios subsequentes.
- Estático: Refere-se a se o ambiente muda ao longo do tempo.
- Discreto: Indica se o espaço de ações ou estados do agente é discreto, o que significa que existem um número finito e bem-definido de opções.
- Multiagente: Indica se existem múltiplos agentes interagindo no mesmo ambiente

Agente	Observável	Determinístico	Episódico	Estático	Discreto	Multiagente
Robô	Sim	Não	Sim	Sim	Sim	Não

Table 1. Modelagem das características do Sistema

Compreendendo as características do ambiente, é possível iniciar a codificação. Para facilitar os testes, foi desenvolvida uma ferramenta em python utilizando uma biblioteca de interface gráfica que gera HTML chamada ”streamlit” juntamente com uma biblioteca para plotagem de gráficos chamada plotly. A interface em HTML é responsável pelas parametrizações do sistema . A figura 1 representa o menu da ferramenta.

As parametrizações possíveis são as seguintes:

- Quantidade de Execuções, representando quantas vezes vamos rodar um experimento
- Tipo de Codificação, representando o tipo do indivíduo, para este problema em específico apenas a codificação inteira está disponível.
- Tipo de Seleção, aqui possuimos a opção de roleta e torneio estocástico para escolha
- Tipo de Crossover, a ferramenta comporta o crossover de ”1 Ponto”, ”2 Pontos” e ”Unificado”
- Tipo de Mutação, que para ese problema em específico a ferramenta apenas disponibiliza o ”Valor aleatório no domínio”
- Elitismo, uma opção booleana responsável pela decisão de preservar o melhor indivíduo de cada geração, ou não.
- Dimensão dos Indivíduos, responsável pelo tamanho do cromossomo e quantidade de genes de um indivíduo
- Tamanho da População Inicial, quantos indivíduos farão parte da população
- Quantidade de Gerações, quantos loops evolutivos teremos

Tendo em mente todas estas características do sistema, podemos mergulhar ainda mais fundo na resolução do problema e de fato entender como que o robô consegue encontrar um caminho até a saída.

A figura 2 exemplifica como funciona a movimentação do robô e cálculo da fitness, temos um indivíduo representado por um cromossomo, e o robô posicionado no início do labirinto, o primeiro passo do algoritmo é descobrir quais são os movimentos possíveis a partir da posição atual, uma estrutura de repetição rapidamente realiza essa

operação e para o caso da imagem retorna um array de tamanho um contendo a posição ”Baixo”. Como é o primeiro movimento do robô, usaremos o primeiro gene para selecionar o movimento que faremos, isso é feito pegando o valor do gene e calculando o resto da divisão deste valor pela quantidade de movimentos possíveis, e o resultado será o índice no vetor de movimentos que iremos realizar, neste caso, como o vetor só possui tamanho um, o único movimento possível seria ”Baixo”. Para a segunda movimentação, temos um vetor de movimentos de tamanho 2, com ”Baixo” e ”Direita”, e um gene com valor 3, portanto o resto da divisão de 3/2 retorna 1, fazendo com que o robô escolha ir para a Direita, visto que é a posição 1 do vetor. Além disso, é utilizada a distancia de manhattan para definir a distância da posição atual do robô até o final do labirinto, e assim convergir para soluções com menor distância.

3. Labirinto 30x25

O primeiro caso que vamos testar nossa solução, é um labirinto de tamanho 30x25, as parametrizações utilizadas foram as seguintes:

- Quantidade de Execuções: 10
- Codificação: Inteiro entre 0 e 3
- Seleção: Torneio Estocástico
- Crossover: 1 Ponto
- Mutação: Valor Aleatório no Domínio
- Elistimo: True
- Dimensão dos indivíduos: 100
- Tamanho População: 10
- Quantidade de Gerações: 1000

O gráfico de convergência da figura 3 mostra a convergência de todas as 10 execuções em diferentes cores, com uma linha tracejada representando a média de todas. Analisando as curvas conseguimos perceber que muitas convergiram para uma solução ótima (saída do labirinto) e algumas não conseguiram convergir e ficaram presas em um sub-ótimo. A tabela 3 apresenta os resultados obtidos, conseguimos perceber que 7 das 10 execuções alcançaram a saída, enquanto as outras 3 ficaram presas em um ponto bem ao lado da saída, que devido sua baixa distância para a saída acabou se tornando um bom atrator local. As figuras 4 e 5 mostram, respectivamente, o caminho percorrido pela melhor solução encontrada, e o caminho percorrido pela pior solução que ficou presa a uma posição próximo da saída

Execução	**Movimentos**	**Chegou na saída?**
Exec 1 e 7	66	Sim
Exec 2	62	Sim
Exec 3 e 4	68	Sim
Exec 5	64	Sim
Exec 6	68	Não
Exec 8	72	Sim
Exec 9 e 10	74	Não

Table 2. Resultados Obtidos

Figure 1. Tela Inicial Ferramenta de aplicação do algoritmo evolutivo

4. Conclusão e Trabalhos Futuros

A implementação desta ferramenta contribuiu para consolidação de todos os conceitos básicos de computação evolutiva, vistos de forma teórica durante as aulas ministradas e artigos lidos. Permitiu perceber o quanto é util a utilização de aplicações de computação evolutiva principalmente na resolução de problemas complexos. Um ponto que foi bastante crítico e que atacou diretamente a qualidade dos testes foi o paralelismo, a ferramenta não foi desenvolvida de forma paralela ao aplicar a função fitness em todos os casos, causando um alto custo computacional e uma certa lentidão ao realizar os testes. Então como trabalhos futuros seria bastante interessante implementação de técnicas de paralelismo pra otimização da complexidade de tempo do código. Outro ponto interessante, seria realizar o teste em labirintos maiores e talvez com mais complexidade ainda, inserindo por exemplo armadilhas no trajeto que adicionam penalidade na trajetória, e também testar outras codificações, por exemplo uma abordagem utilizando uma codificação Real.

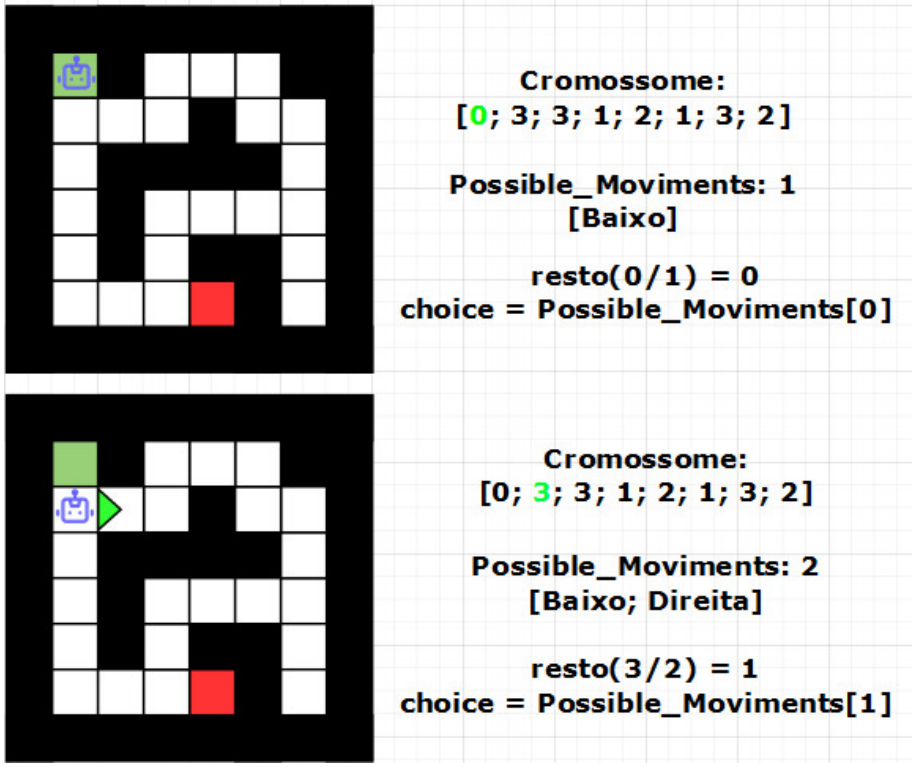

Figure 2. Movimentação do robô

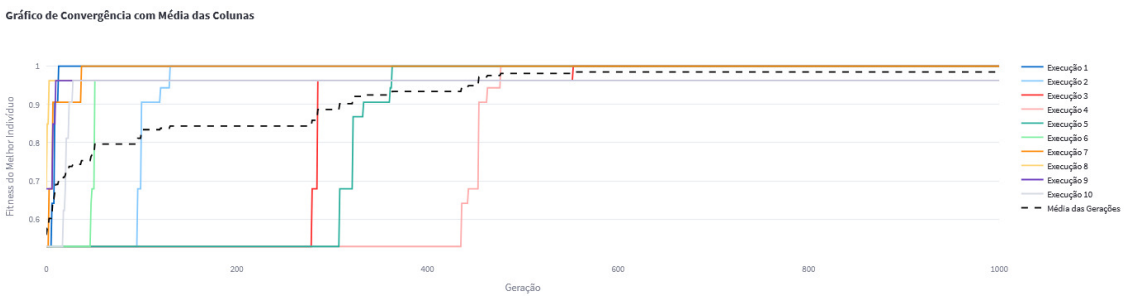

Figure 3. Gráfico de Convergência

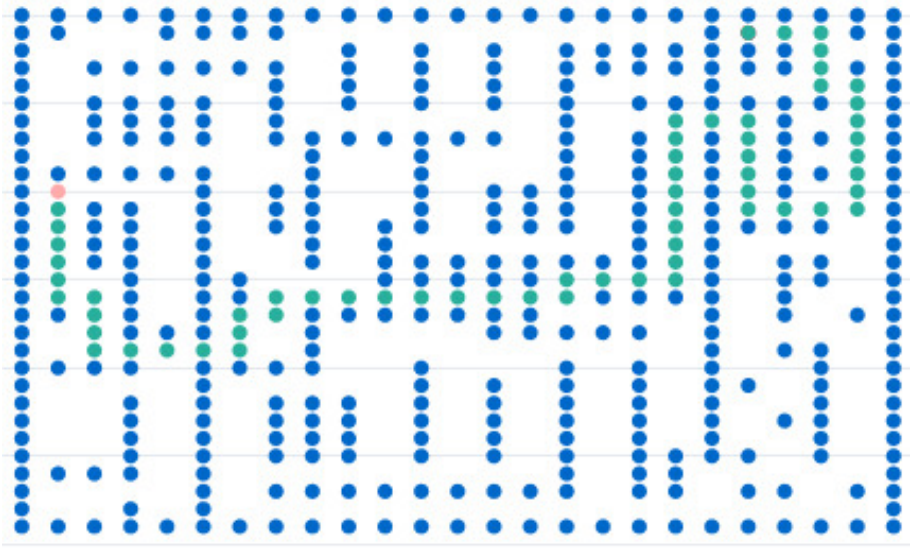

Figure 4. Caminho realizado por Execução 2

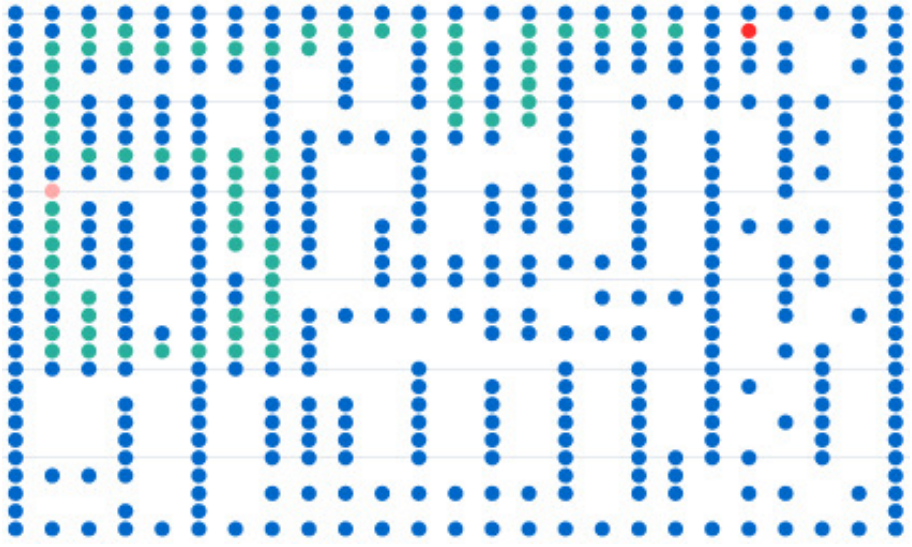

Figure 5. Caminho realizado por Execução 9