Cleaner Robot: aplicação do algoritmo genético ao problema de rota do robô aspirador

Thiago Santos Da Silva. RA: 251676.
Laura Lopes Carneiro. RA: 251280.

MS 428

Resumo

Com o objetivo de aprimorar a tecnologia de elaboração de rotas percorridas por robôs de limpeza, este trabalho busca testar a viabilidade e a eficácia da aplicação de modelos de algoritmos genéticos a esse problema específico. [1]

1 Introdução

Utensílios domésticos modernos têm sido cada vez mais usados para nos ajudar com as tarefas do dia a dia. Dentre eles, destaca-se o robô aspirador, já bastante difundido no mercado. No entanto, esse aparelho ainda possui grande margem para melhorias, principalmente no que diz respeito a desafios de navegação e superfícies irregulares [1].

Neste projeto, focamos na questão da navegação, buscando otimizar a rota percorrida pelo robô aspirador, aumentando sua eficiência, diminuindo o gasto de bateria e, ao mesmo tempo, levando em consideração os obstáculos da superfície, a capacidade do depósito de lixo do aspirador e os locais em que o lixo pode ser esvaziado.

Com essa finalidade, desconsideramos questões relacionadas à detecção de objetos, ao processamento de imagem e à leitura do ambiente, focando nossa análise na etapa da limpeza. Portanto, nosso problema aproxima-se de um problema clássico de pesquisa operacional chamado de *Problema do caixeiro viajante* (TSP), que consiste em encontrar a melhor rota possível, visitando todos os pontos de um conjunto de localizações uma única vez.

No caso que estudamos, a melhor rota é a de menor custo, ou seja, a de menor distância, reduzindo, assim, o gasto de bateria e o tempo de limpeza do ambiente, desde que sejam respeitadas as restrições do ambiente e do robô aspirador.

A dificuldade encontrada quando lidamos com esse tipo de problema é que o número de soluções cresce exponencialmente com o aumento da quantidade de destinações, o que eventualmente pode superar a capacidade de processamento até mesmo dos computadores mais potentes. Assim, utilizar métodos baseados em "força bruta", que escolhem a melhor solução dentre todas as possíveis, torna-se inviável quando pensamos em pequenos equipamentos, como os robôs de limpeza, de baixo poder computacional [2].

Desse modo, a alternativa proposta neste trabalho é a utilização de computação evolucionária baseada em modelos de algoritmos genéticos para gerar soluções que resolvam as demandas propostas. Nesse caso, não necessariamente encontramos a solução ótima do problema; no entanto, podemos obter soluções razoáveis de forma muito mais eficiente, o que se encaixa com as restrições e capacidades do robô aspirador.

Algoritmos genéticos pertencem à classe de algoritmos heurísticos. Isto é, não há garantia de que sua solução irá convergir para a solução ótima, e a velocidade de convergência depende de hiper fatores. Apesar de existir pouca literatura a respeito

[1] O código feito para o problema estudado pode ser encontrado em: GitHub - otimização de rota do robô aspirador.

da escolha de parâmetros ideias para cada problema, esse método se popularizou por apresentar bom desempenho na prática.

Outra vantagem dos algoritmos genéticos é a garantia de que alguma solução será encontrada, mesmo que ela ainda não seja boa o suficiente, o que nos permite estabelecer um parâmetro a partir do qual uma solução seria considerada aceitável. Assim, é possível diminuir o tempo de espera para encontrar uma solução, o que é essencial devido à natureza do produto, já que é esperado que o robô aspirador comece a limpar um cômodo assim que for ligado e, além disso, pode haver mudanças no ambiente que devem ser levadas em consideração, levando a um novo cálculo de rota, o que deve ser um processo rápido e eficiente.

Um algoritmo genético pode ser resumido, de forma genérica, a partir dos seguintes passos:

- 1)inicialização da população de cromossomos;
- 2)avaliação de cada cromossomo;
- 3)seleção de pais que gerarão novos filhos.
- 4)recombinação e mutação.
- 5)velhos membros da população são apagados e novos cromossomos são avaliados e inseridos na população.
- 6)Se o tempo acabou ou a solução é aceitável de acordo com o parâmetro escolhido, o melhor cromossomo é retornado. Caso contrário, voltamos para a etapa 3) [3].

O modelo esquematizado pode ser visto na Figura 2. Nas próximas seções, explicaremos em maior detalhe como esse modelo foi aplicado, em específico, ao problema estudado.

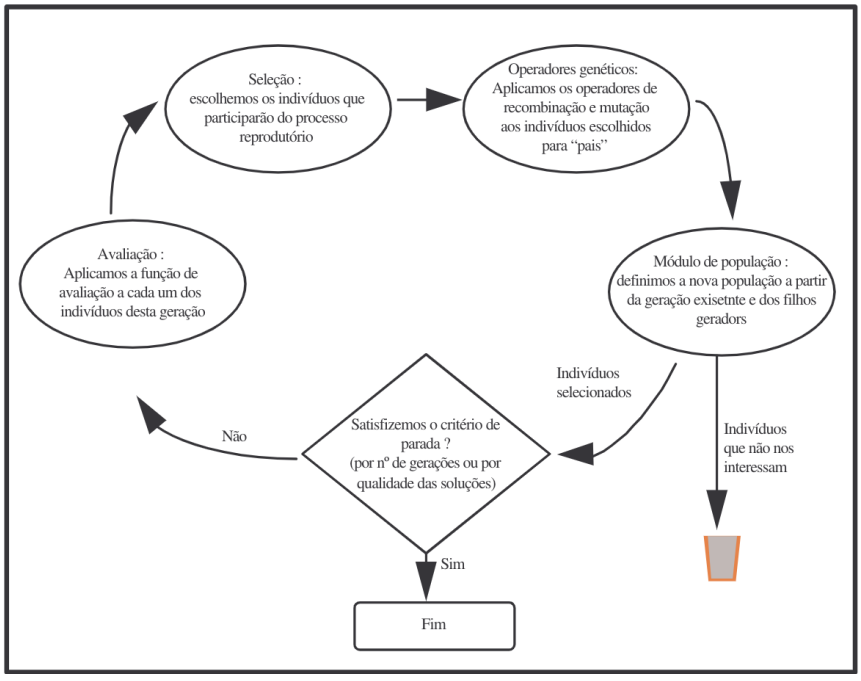


Figura 2: Modelo esquematizado de um algoritmo genético [3].

2 Descrição do Problema

A fim de modelar o problema de rota do robô aspirador, consideramos uma malha formada por dois tipos de pontos: aqueles que representam superfícies pelas quais o robô pode passar livremente, e aqueles onde a passagem está bloqueada. Além disso, nas extremidades da superfície, também há alguns pontos especiais, que representam os depósitos de lixo, onde o robô obrigatoriamente deve ir despejar seu estoque toda vez que estiver cheio.

Ademais, adotamos as seguintes premissas:

- A bateria dura um tempo muito maior do que o estoque de sujeira. Portanto, para todos os fins do problema, ela é considerada infinita;
- O robô sempre iniciará e terminará um caminho em um ponto de descarte de lixo, mesmo que ele não tenha atingido a capacidade máxima do estoque;
- Uma solução só é válida se toda a área foi limpa;
- Os pontos de descarte de lixo sempre ficam na borda da superfície que deve ser limpa;
- A quantidade de lixo acumulada durante o caminho deve ser menor ou igual ao estoque máximo do robô;
- O robô irá limpar todo o espaço de forma homogênea.

Nosso objetivo é minimizar o número de pontos percorrido pelo robô; minimizando, assim, o tempo gasto para realizar a limpeza. Desse modo, nosso problema de minimização pode ser escrito da seguinte forma:

Seja x_{ik} o ponto i do caminho k.

Seja P_{ijk} a variável de decisão, que indica se fomos do ponto i ao j no caminho k.

Seja e_{ijk} a quantidade de lixo que entra no estoque ao passarmos do ponto i ao ponto j pelo caminho k.

Seja x^d_{ij} um ponto especial na malha, em que podemos realizar o descarte de resíduo, ou seja, o depósito de lixo.

Então, temos o seguinte modelo:

$$\min \sum_{i,j,k} p_{ijk}$$

$$\begin{cases} \sum_{i,j} e_{ijk} \leq e_{max} \quad \text{(limite de estoque do robô)} \\ x_{i_{max},j_{max},l} = x_{i_0,j_0,l+1} = x^d_{ij} \text{ (ponto final de um caminho é inicial de outro; continuidade)} \\ \sum_k p_{ijk} \geq 1, \quad \text{(passa pelo menos uma vez em cada ponto)} \end{cases}$$

Assim, o algoritmo fica responsável por escolher em quais pontos o robô irá parar para descartar os resíduos; ao mesmo tempo, percorrendo o menor caminho.

3 Descrição do modelo aplicado

Utilizamos a linguagem de programação **Python** para construir o código que soluciona nosso problema, por ser uma linguagem que facilita trabalhar com orientação a objeto e por ser muito utilizada atualmente.

Com essa finalidade, criamos duas classes principais, **ClearRobot** e **Soluctions**. A primeira contém as principais funções associadas ao robô físico: a superfície que ele deve limpar e os caminhos que ele pode percorrer nessa superfície. Enquanto a segunda possui funções relacionadas às propriedades esperadas de uma solução do problema; dentre elas, ser factível e ter um valor de função de avaliação. Assim, no modelo de algoritmo genético, as possíveis soluções, correspondentes aos possíveis caminhos percorridos, representam nossos indivíduos, que serão julgados por meio do valor de função de avaliação. Nesse caso, a função de avaliação é equivalente à função objetivo, pois consideramos que o caminho de menor distância é o melhor caminho.

Como exemplo de representação de superfície que deve ser limpa pelo robô, podemos visualizar a Figura 3 descrita como uma malha $10_{\times}10$, onde os números "1" representam pontos por onde o robô pode passar, "0", pontos de acesso impedido e "2", depósitos de lixo, onde o robô obrigatoriamente deve ir quando seu estoque estiver cheio:

0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0

Para aprimorar a visualização da superfície, podemos representá-la por meio de figuras. Portanto, como pode ser visto na Figura 3, os pontos "1"correspondem à cor verde-água, "0", à cor roxa e "2", ao amarelo.

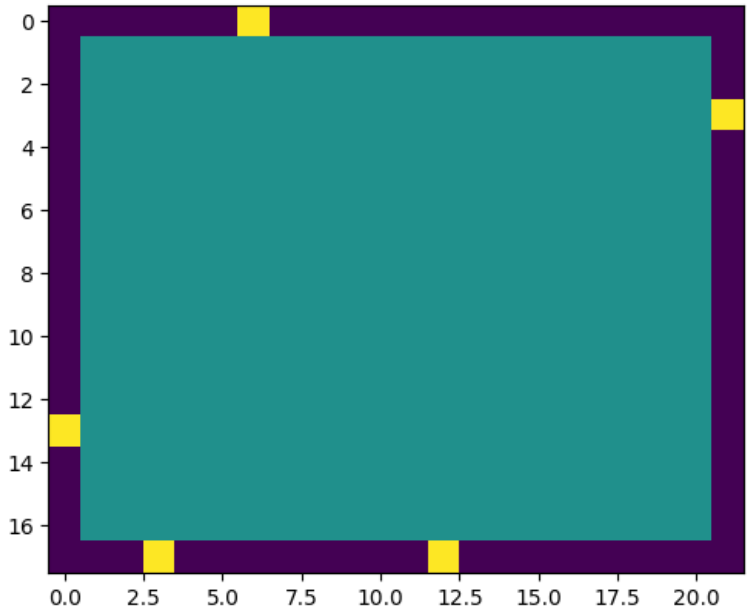
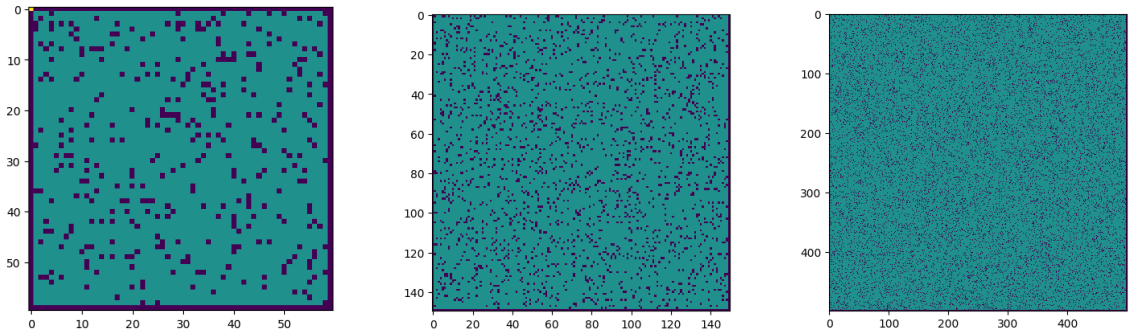


Figura 3: Exemplo de superfície.

A fim de facilitar a criação de exemplos de superfícies, foi feito um método capaz de traçar qualquer malha de forma aleatória, seguindo o modelo de superfície descrito. Na Figura 4, podem ser visualizados três desses exemplos.



((a)) Exemplo 60× 60. ((b)) Exemplo 150 × 150. ((c)) Exemplo 500 × 500.

Figura 4: Exemplos de três malhas (superfícies) de diferentes tamanhos criadas de forma aleatória.

Nessas superfícies, cem diferentes caminhos serão traçados de forma aleatória, e irão "competir"entre si, geração a geração, até encontrarmos uma solução boa o suficiente para o problema. Assim, a cada "geração"de caminhos, temos cem indivíduos. Optamos por esse número por ele não ser gigantesco a ponto de aumentar demais os custos computacionais, mas ser grande o suficiente para obtermos uma heterogeneidade de caminhos, evitando o risco de não termos nenhum caminho bom na população. Dentre eles, alguns irão ser selecionados para "se reproduzir", ou seja, dois caminhos-pais são selecionados, por meio do mecanismo da *roleta viciada*, para criar caminhos-filhos, gerados pelo operador de *crossover*, até formarmos uma nova geração, novamente com cem indivíduos (caminhos). Seria possível alterar

o tamanho da próxima geração, por meio do parâmetro *population*; nesse caso, o tamanho da geração seguinte é o tamanho da geração atual multiplicado por esse parâmetro. No entanto, optamos por deixar *population*=1, de modo que cada geração possua o mesmo número de indivíduos (cem). Além disso, cada vez que esse processo de geração de novos caminhos ocorre, pode atuar sobre ele o operador *mutação*, que irá modificar radicalmente a solução. Ademais, alguns indivíduos da geração anterior podem passar para a próxima geração intactos, sem sofrer seleção, por meio do operador de *elitismo*.

Cada um desses processos será explicitado em maiores detalhes nas subseções seguintes.

3.1 Operador de Seleção

Para realizar a competição entre as soluções, foi escolhido o método da *roleta viciada*. Nele, os pais mais capazes, ou seja, os caminhos mais bem avaliados de acordo com a nossa função, têm maior probabilidade de gerar filhos, mas isso não impede que os pais menos aptos também gerem descendentes. Isso ocorre porque, mesmo que caminhos possuam um valor ruim, pedaços dele podem ser úteis para gerar um excelente caminho. Portanto, não podem ser excluídos. Com essa finalidade, é feita uma roleta (virtual) na qual cada caminho recebe um pedaço proporcional à sua avaliação [3]. No entanto, como o objetivo, em nosso estudo, é minimizar o valor do caminho, o pedaço recebido por cada um é inversamente proporcional a seu valor de avaliação. Portanto, privilegiamos indivíduos com número mais baixo de função. Um exemplo de *roleta viciada* pode ser visto na Figura 5. Nota-se, a partir disso, que esse é um método baseado em fatores probabilísticos.

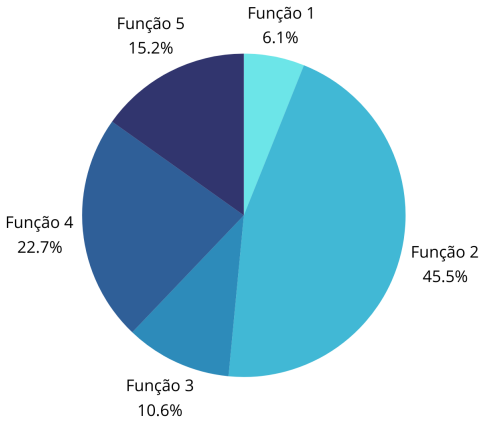


Figura 5: Exemplo da roleta criada a partir do valor de função de avaliação de cada indivíduo.

3.2 Operador de CrossOver

O *crossover* é o operador de recombinação. A partir dele, dois caminhos-pais dão origem a dois caminhos-filhos. Ele é responsável por realizar um corte no indivíduo e recombiná-lo com outro. No problema estudado, escolhemos um ponto em comum em que o aspirador passa várias vezes para defini-lo como ponto de corte. Assim, a parte posterior a esse corte no caminho é concatenada à parte anterior de

outro caminho, e vice-versa. A Figura 6 mostra uma ilustração desse processo: antes do *crossover*, havia dois indivíduos, um roxo e um rosa; após a recombinação, há dois novos indivíduos, formados por partes dos anteriores.

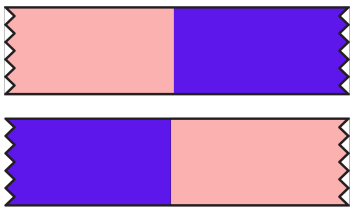


Figura 6: Ilustração de como novos caminhos são gerados (*crossover*).

Desse modo, caso as novas soluções sejam infactíveis, escolhemos um novo plano de corte e refazemos o processo. Se forem factíveis, adicionamos à geração atual.

3.3 Operador de Mutação

O operador de *mutação* entra em cena depois de composto o filho e possui uma chance baixa de ocorrer. Nesse caso, escolhemos 0,3% de ocorrências. Sua função é gerar variação dentro do espaço de soluções (caminhos-filhos), evitando que as soluções convirjam para um mínimo local e possibilitando a convergência mais rápida para uma solução boa que esteja distante dos caminhos-pais. Em nosso estudo, a *mutação* ocorre removendo o interior de um caminho a partir da escolha de pontos de corte. Assim, o valor de função do caminho irá diminuir; no entanto, pode também tornar-se infactível, o que a inviabilizaria de compor a nova geração. A Figura 7 apresenta uma ilustração desse processo, que atua removendo a parte em azul do indivíduo.

Figura 7: Ilustração do operador de mutação.

3.4 Operador de Elitismo

Esse operador ajuda a garantir que a performance do algoritmo esteja sempre crescendo com o decorrer das gerações [3]. Seu funcionamento ocorre da seguinte forma: um certo número de indivíduos da população, os mais bem avaliados, passam para a próxima geração intactos. Em nosso trabalho, os "melhores"são os caminhos de menor tamanho, ou seja, de menor valor de função. Na Figura 8, as soluções são mostradas enumeradas de acordo com seus valores de função. Nesse caso, as primeiras soluções seriam as escolhidas a passar para a próxima geração, pois possuem menores valores de função.

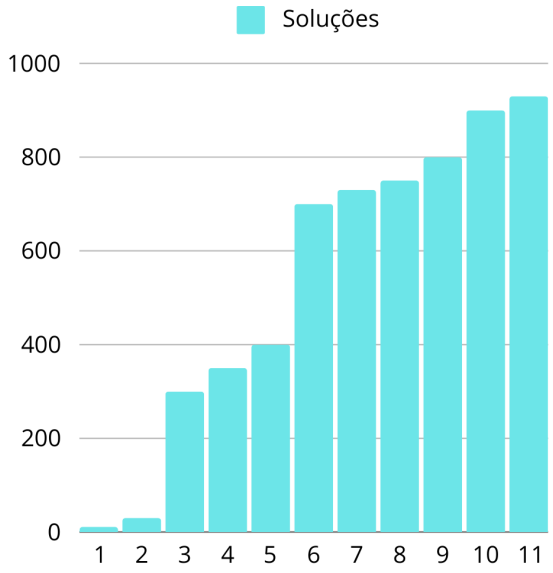


Figura 8: Exemplo da enumeração das soluções a partir de seus valores de função.

4 Resultados

4.1 Caso 2x2

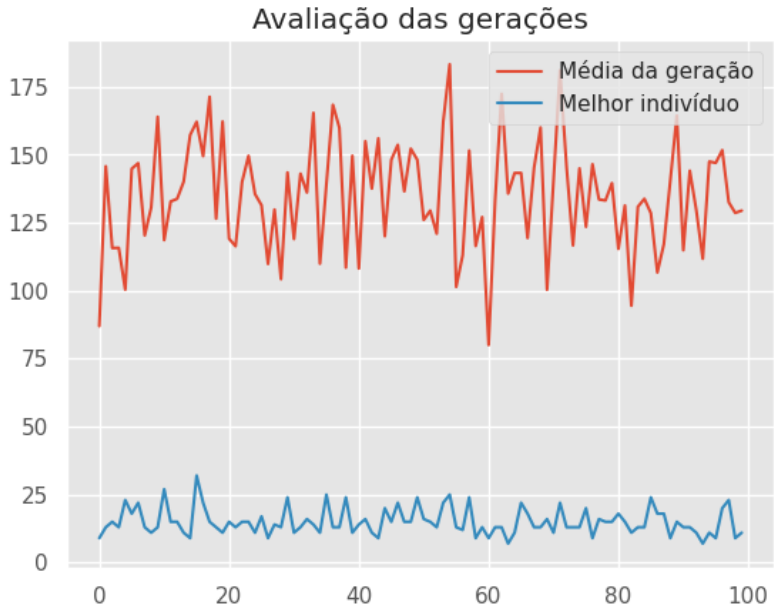


Figura 9: Avaliação das gerações em malha 2x2.

Esse caso é o mais simples e rápido. Consideramos 50 soluções competindo a cada geração, com parâmetro *elitismo* de 3 indivíduos.

Foi possível criar uma grande variedade genética e o melhor indivíduo (caminho) permaneceu através das gerações. Dessa forma, o valor da solução permanece

relativamente estável, como pode ser visto na Figura 9. No entanto, notamos que a média do valor dos caminhos das vinte primeiras gerações aumentou, ou seja, as soluções pioraram nesse período.

4.2 Caso 4x4

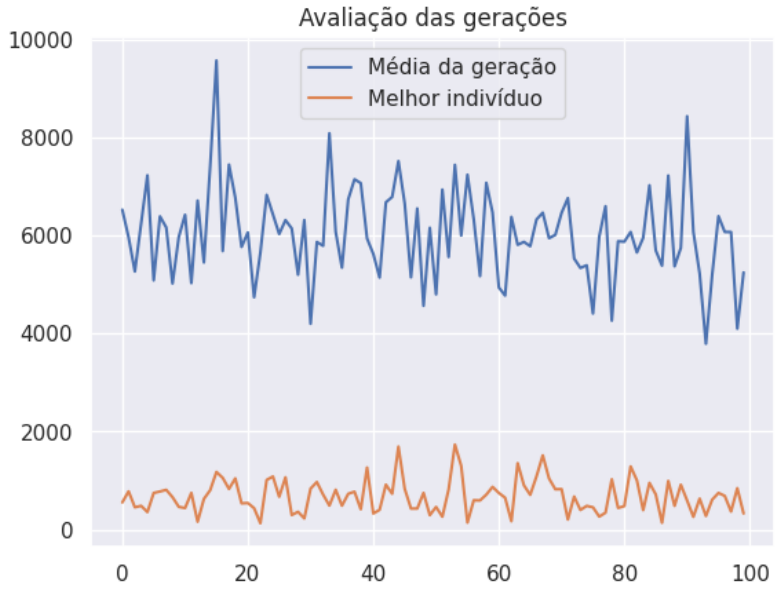


Figura 10: Avaliação das gerações em malha 4x4.

Como mostra a Figura 10, percebemos que o valor médio das gerações é 60 vezes maior do que o caso 2x2. Isso se deve ao modo como estruturamos o código, permitindo grandes soluções. No entanto, a média da população diminui.

Um grande empecilho que limitou o trabalho foi o poder computacional necessário, visto que, ao construir uma malha grande, as chances de um caminho aleatório passar por um ponto em específico diminuem. Uma saída é condicionar a criação de caminhos à redução de lixo, fazendo com que cada caminho obrigatoriamente tenha que reduzir o lixo da malha geral.

5 Conclusão

Ao longo dos testes, usamos poucas gerações e já foi possível observar certa convergência. Assim, não foi observado ponto de estagnação nas figuras 9 e 10. Na primeira imagem, a convergência se deu por ser um problema pequeno e, na segunda, ficou mais clara a melhora da população.

No entanto, ao analisarmos o desempenho do algoritmo, fica evidente que algumas melhorias são necessárias para que ocorra convergência para os melhores caminhos, o que parece não acontecer, já que o robô passa muitas vezes pelos mesmos pontos, inclusive por aqueles onde já havia limpado. Dentre as opções de melhoria, destacamos aumentar a quantidade de gerações e aprimorar a função de *crossover*, que gera novos caminhos, priorizando a criação de caminhos que não possuam tantos pontos repetidos e diminuindo a infactibilidade dos caminhos gerados.

A partir da implementação dessas melhorias, o algoritmo poderá ser generalizado para áreas maiores.

Concluímos que, apesar de muitas melhorias serem necessárias para que o algoritmo se torne aplicável de forma satisfatória a problemas reais, a abordagem do problema do robô aspirador a partir dos algoritmos genéticos mostrou-se promissora e deve continuar sendo estudada e desenvolvida devido a seu potencial de aprimoramento dessa tecnologia.

Referências Bibliográficas

[1] Ramji Jayaram and Rashmi R. Dandge. Optimizing cleaning efficiency of robotic vacuum cleaner.

[2] Marc Kuo. Algorithms for the travelling salesman problem, 2024.

[3] Ricardo Linden. *Algoritmos Genéticos: uma importante ferramenta da Inteligência Computacional.* Brasport, Rio de Janeiro, 2006.