Algoritmos genéticos aplicados a robótica

Gabriela Custódio Martins
Gustavo Storithont Mudri
Pedro Henrique Centenaro

1 Introdução

A evolução é o principio de unificação principal da biologia moderna. A teoria Darwineana clássica juntamente com a seleção de Weismann e a genética de Mendel são a base para a teoria da evolução atual. Em outras palavras, a evolução é o resultado de processos estocásticos interativos (reprodução, mutação, cruzamento e selecção) sobre populações ao longo de gerações. Porém, a teoria da evolução estende-se para além dos sistemas biológicos, quando utilizada por máquinas computadorizadas com o intuito de resolver problemas de otimização. A computação evolutiva é um termo que abrange uma serie de algoritmos entre os quais os algoritmos genéticos (AGs) são um dos ramos principais.

Assim, a base de funcionamento de um AG considera as possíveis respostas geradas pelo problema como indivíduos que competirão entre si pela oportunidade de se reproduzirem. Neste processo, os mais aptos, *i.e.*, que representam uma melhor solução, tem maior chance de perpetuar parte de suas características, aumentando a probabilidade de se obter uma maior adpatação da população em geral. Tem-se como exemplo a saga das mariposas: observou-se um aumento da frequência de mariposas em sua forma escura (denominada carbonaria) em substituição a sua forma clara (typica), por ocasião da poluição causada pela Revolução Industrial do século 19 em Manchester (Inglaterra) e outras cidades industrializadas. Esse fenômeno, chamado melanismo industrial, foi proposto como um exemplo clássico de seleção natural. Devido à cor escura, as mariposas carbonaria se camuflavam na cor de fundo das árvores enegrecidas pelos resíduos industriais. Essa camuflagem as protegia da predação por pássaros. Com isso, tinham uma probabilidade maior de viver até a idade reprodutiva e transmitir seus genes a gerações futuras, aumentando, assim, a frequência dos genes responsáveis pela cor escura [1].

A maneira particular com que os AGs operam faz com que se destaquem características como a adaptabilidade, vista pelo exemplo da saga das mariposas. Visto que a representação e a avaliação das possíveis soluções são as únicas partes (de um considerável conjunto de operações utilizadas em seu funcionamento) que obrigatoriamente requisitam conhecimento dependente do domínio do problema abordado [22], basta a alteração destas para transporta-las para outros casos. Portanto, a preocupação de um programador de AGs não é *de que forma* chegar a uma solução, mas sim *com que ela deveria se parecer*.

2 Visão geral de problemas de otimização

Os problemas de otimização, também conhecidos como problemas de programação matemática, envolvem encontrar um ou mais valores ótimos para um conjunto de expressões matemáticas. Um ótimo pode ser um valor que maximiza uma expressão (máximo) ou que minimiza uma expressão (mínimo). Quando o problema em questão envolve a determinação de máximos, chama-se problema de maximização – ou minimização, no caso de mínimos.

Em cursos de cálculo, os problemas de otimização propostos costumam ser relativamente simples. Geralmente, apresenta-se uma *função objetivo* de uma variável e pede-se que seja maximizada ou minimizada. Neste caso, a resolução costuma ser simples, de acordo com o seguinte passo a passo [32]:

1. Dada a função a otimizar, $f(x)$, determinar as derivadas $f'(x)$ e $f''(x)$.
2. Encontrar os zeros de $f'(x)$, conhecidos como pontos críticos de $f(x)$.
3. Sejam $x_1, \ldots, x_n$ os pontos críticos de f. Calcular $y_k = f''(x_k) \forall k \in \{1, \ldots, n\}$.
4. Se o problema for de maximização, os pontos de máximo são os x_k para os quais $y_k < 0$. Se o problema for de minimização, os pontos de mínimo são os x_k para os quais $y_k > 0$.

Nota-se que podem existir vários pontos máximos e mínimos. É possível, portanto, que x_a e x_b sejam ambos pontos de máximo, com $x_a > x_b$. Isso significa que x_a e x_b resultam nos maiores valores de $f(x)$ em intervalos específicos da função. A Figura 1 exemplifica graficamente esta questão. A função ilustrada contém dois pontos de máximo e um ponto de mínimo. Em relação aos pontos de máximo, o ponto à esquerda está claramente acima de todos os outros pontos da função. Por esta razão, diz-se que este é um máximo *global* da função. Já o ponto à direita, por ser máximo de um intervalo reduzido do domínio da função, é chamado de máximo *local*. Por fim, apesar de a função ter um único ponto de mínimo, este não é um mínimo global, dado que $x \to -\infty \Rightarrow f(x) \to -\infty$ e $x \to \infty \Rightarrow f(x) \to -\infty$. Ou seja, é possível que um problema não tenha ótimos globais.

Problemas reais de otimização costumam ser mais complexos do que isso. Um exemplo clássico considera uma empresa que precisa decidir qual quantidade de cada produto produzir para maximizar seu lucro [34]. Neste caso, não faz sentido qualquer solução onde a quantidade de determinado produto seja < 0 e, portanto, é necessário impor *restrições* ao modelo do problema. Também é importante observar que n produtos

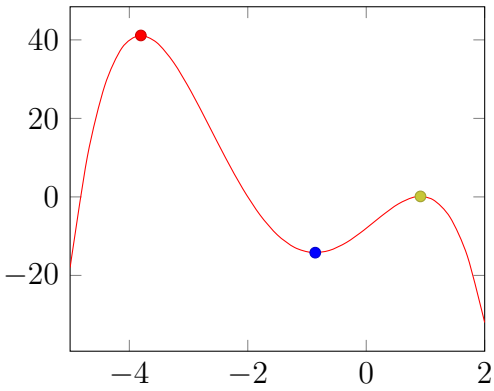

Figura 1: Pontos de máximo e mínimo de uma função

diferentes representam n quantidades diferentes que devem ser definidas para resolver o problema de otimização. Ou seja, a função objetivo é uma função de n variáveis.

Todos os fatores apresentados até aqui conspiram contra a resolução do problema por métodos manuais. De fato, a maioria dos problemas de otimização tem sua resolução feita por métodos computacionais, como o método simplex [6, 7], comumente utilizado para resolver problemas de programação linear – ou seja, problemas cuja função objetivo e cujas restrições são lineares. Entretanto, mesmo o simplex tem suas limitações. Em trabalho de iniciação científica realizado sobre este método [10], constatou-se que os principais fatores que dificultam a resolução de problemas são:

- Grande número de restrições: A resolução de problemas de otimização por meio do método simplex é baseada numa interpretação geométrica das restrições impostas sobre o modelo. Assim, quanto maior o número de restrições, mais complexa é a geometria do problema, o que pode resultar em mais passos para resolvê-lo.

- Restrições inteiras: Pode-se querer que certas variáveis do problema sejam números naturais, inteiros ou binários. Neste caso, o método simplex deixa de funcionar, e costumam-se aplicar métodos de otimização inteira como o *branch-and-bound*, que cria uma árvore de sub-problemas que podem ser resolvidos usando o simplex. Tal árvore pode crescer rapidamente, executando o simplex várias vezes, sem a garantia de convergência para um valor ótimo em cada galho.

No caso de problemas não-lineares, a otimização é ainda mais difícil, pois existem vários métodos diferentes, que têm vantagens e desvantagens dependentes de características matemáticas específicas de cada problema [26].

2.1 Heurísticas para otimização

Métodos como o simplex são ditos *exatos*, pois são feitos para retornar valores ótimos de fato. Contudo, como arguido anteriormente, problemas muito complexos podem ter uma resolução exata muito lenta. Por este motivo, muitos problemas práticos de otimização são resolvidos por *heurísticas*. As heurísticas são métodos que visam simplificar modelos de otimização de modo a resolvê-los mais rapidamente. Frequentemente, isso resulta em soluções não-ótimas, de onde resulta que uma "boa" heurística deve ser rápida e, ao mesmo tempo, conduzir a valores próximos do ótimo. Convém observar que heurísticas costumam ser separadas em duas categorias, que são apresentadas a seguir.

Heurísticas convencionais

Heurísticas convencionais são projetadas para resolver problemas específicos. Um exemplo é a heurística de George e Robinson [17] para carregamento de contêineres com itens cuboides. Para concluir este objetivo, a heurística fatia o contêiner em vários volumes cuboides menores e aplica uma série de procedimentos para preenchê-los. Uma heurística popular para resolver problemas de roteamento de veículos é o método *nearest neighbor* [21], que consiste em construir rotas ponto a ponto, sempre escolhendo o ponto não visitado menos distante como próximo ponto do trajeto. Como estas heurísticas criam soluções do zero, são chamadas de heurísticas *construtivas*. Geralmente, heurísticas construtivas são aperfeiçoadas por heurísticas *de melhoria*, que costumam introduzir alterações aleatórias nas soluções geradas em busca de soluções melhores.

Meta-heurísticas

Em geral, as meta-heurísticas operam em uma de duas maneiras. No primeiro caso, o método extrai informações específicas sobre o problema e em seguida escolhe uma heurística que parece mais apropriada para resolvê-lo. No segundo caso, a meta-heurística é baseada em processos naturais que podem ser abstraídos para resolver uma vasta gama de problemas. É possível citar métodos como recozimento simulado (*simulated annealing*), algoritmos genéticos, colônia de formigas, busca tabu, entre outros. Como exemplo, o primeiro procedimento citado é descrito a seguir.

O recozimento simulado é baseado numa técnica metalúrgica de remoção de tensões internas provenientes de deformidades na estrutura cristalina de um metal. De acordo com Delahaye, Chaimatanan e Mongeau [14], o processo consiste em aquecer o metal até seu ponto de fusão. Neste estado, os átomos que constituem o metal estão dispersos

aleatoriamente no espaço. Em seguida, inicia-se um processo lento de resfriamento, após o qual os átomos tendem a se distribuir de forma homogênea, de modo a minimizar a energia retida na estrutura cristalina.

Para imitar os procedimentos descritos, o recozimento simulado precisa de uma função (f) capaz de avaliar soluções para o problema especificado, o que é análogo a medir a energia da estrutura cristalina do metal. No lugar de tempo, usa-se o número de iterações do método (k) para "cronometrar" o processo. Além disso, introduz-se um parâmetro de temperatura (T) que é reduzido a cada iteração. O método costuma se resumir aos seguintes passos:

1. Gerar uma solução inicial e inicializar os parâmetros.
2. Gerar uma nova solução, fazendo modificações na solução atual.
3. Comparar a solução nova à atual usando a função f.
4. a) Se a solução nova for melhor do que a atual, ela passa a ser a solução atual.
   b) Se a solução nova for pior do que a atual, utiliza-se um método aleatório para determinar se ela deve substituir a atual, de modo que as chances sejam maiores quando a temperatura T for maior.
5. Reduzir a temperatura T.
6. Se a temperatura T tiver chegado ao seu valor mínimo, terminar o algoritmo e retornar a solução atual. Do contrário, voltar ao passo 2.

3 Algoritmos genéticos

Algoritmos genéticos (AGs) são uma meta-heurística baseada em princípios de seleção natural para obtenção de boas soluções para problemas. As características dos AGs os tornam excelentes ferramentas para vários propósitos diferentes. Nesta seção, são descritos os passos que todos os AGs devem seguir, e são apresentadas aplicações destes algoritmos.

3.1 Estrutura de algoritmos genéticos

Os AGs simulam mecanismos evolutivos para obter soluções para problemas. Para tal, os AGs dependem de uma operação de conversão de soluções para cromossomos e vice-versa. Um cromossomo é uma estrutura de dados que contém as informações necessárias para reconstruir e avaliar uma solução. Geralmente, utilizam-se vetores de números para representar um cromossomo. Um exemplo simples de cromossomo é descrito por Hermawanto [19], que propõe um AG para resolução de um problema de minimização de quatro variáveis. Para resolver este problema, o autor representa os cromossomos como vetores de quatro números inteiros, cada um correspondendo ao valor de uma variável. Em problemas mais complexos, a conversão cromossomial pode ser mais complicada. No caso do exemplo citado, como os genes informam diretamente os valores das variáveis do problema, diz-se que a informação necessária para decodificar o cromossomo está no seu *genótipo*. No entanto, se os valores nos genes precisarem passar por algum processo de "tradução" para corresponderem a informações concretas, diz-se que as informações das variáveis estão contidas no *fenótipo* do cromossomo [16].

Após a determinação do processo de conversão cromossomial supracitado, é necessário determinar uma função de aptidão (*fitness*), ou seja, uma função que retorne valores através dos quais seja possível comparar cromossomos. Na realidade, tal função não precisa ser matematicamente objetiva - o processo de classificação de uma solução pode ser subjetivo [29], o que significa que AGs podem ser utilizados para resolver problemas de otimização de caixa preta (*blackbox*), ou seja, problemas cuja função objetivo ou restrições não são conhecidas ou bem-definidas [2]. Feito isso, resta implementar os passos seguintes, cuja descrição é baseada, principalmente, na obra de Sastry, Goldberg e Kendall [29].

3.1.1 Inicialização

Na etapa de inicialização, as primeiras soluções para o problema são geradas. Tais soluções podem ser completamente aleatórias ou advir de heurísticas de construção espe-

cializadas. Uma diferença importante entre os AGs e as técnicas de recozimento simulado (RS) está no fato que o RS mantém uma única solução ao longo de todo o processo de resolução do problema, ao passo que nos AGs existem $n \geq 2$ soluções. O conjunto de soluções é chamado de *população*, e traz consigo a vantagem de explorar, concorrentemente, várias alternativas diferentes de resolução de problema (a este respeito, o trabalho de Murawski e Bossaerts [24] é bastante elucidativo, mostrando como vários agentes, utilizando diferentes heurísticas, descobrem uma parte muito maior do conjunto solução de um problema do que agentes individuais).

Apesar do poder de exploração da população, é necessário escolher o seu tamanho com cautela. Populações muito pequenas podem apresentar baixa variedade cromossomial e, consequentemente, convergir para ótimos locais ao longo do processo evolutivo. Por outro lado, populações muito grandes podem ser computacionalmente ineficientes, se populações menores forem capazes de atingir os mesmos resultados [27]. Para problemas com custo computacional muito elevado, Delahaye, Chaimatanan e Mongeau [14] recomendam a utilização de métodos sem população, como o RS.

3.1.2 Avaliação

Na avaliação, os cromossomos da população atual são avaliados pela função de aptidão. Como abordado anteriormente, esta função pode ser objetiva ou subjetiva - o importante é que, ao fim do processo, os cromossomos possam ser comparados. Quando o valor da função de aptidão é objetivo, é necessário especificar o que faz de um valor melhor ou pior do que o outro. Por exemplo, no AG de Hermawanto [19] para resolução de problemas de minimização, a função de aptidão é a própria função a minimizar e, portanto, soluções que obtiverem valores menores serão melhores. Se o problema a resolver fosse de maximização, valores maiores seriam melhores.

3.1.3 Seleção

O processo de seleção leva em conta os resultados da etapa de avaliação para definir casais (pares) de cromossomos. Os métodos de seleção costumam implementar processos estocásticos em que os cromossomos com avaliações melhores são favorecidos. Na sequência, são abordados alguns métodos.

Método da roleta: Sejam $n \geq 2$ o tamanho da população e q_i $(i = 1, \ldots, n)$ os valores de aptidão de cada solução. A cada solução i é associada uma probabilidade $p_i = q_i / \sum_{j=1}^{n} q_j$ de que i seja selecionada. À solução 1 fica atribuído o intervalo $I_1 = [0, p_1]$. A partir disso, é possível determinar os intervalos das soluções $k = 2, \ldots, n$, em sequência,

com $I_k = \left(\sum_{j=1}^{k-1} p_j, \sum_{j=1}^{k} p_j\right]$. Evidentemente, $\cup_{j=1}^{n} I_j = [0, 1]$, então, gera-se um valor aleatório $r \in [0, 1]$, de modo que $r \in I_k$ significa que a solução k deve ser selecionada [9, 29].

A título de exemplo, supõe-se uma população de cinco soluções, cujos dados são apresentados na Tabela 1.

Tabela 1

Solução (k)	Aptidão (q_k)	Probabilidade (p_k)	Intervalo (I_k)
1	4.52	0.156	[0, 0.156]
2	9.04	0.313	(0.156, 0.469]
3	7.33	0.254	(0.469, 0.723]
4	3.21	0.111	(0.723, 0.834]
5	4.79	0.166	(0.834, 1]

Método de *ranking*: Neste método, ordenam-se as soluções da pior para a melhor, baseado em seus valores de aptidão. Em seguida, a cada solução k é atribuido um valor r_k equivalente à posição da solução na lista ordenada. Desta forma, a pior solução tem $r_k = 1$, e a melhor solução tem $r_k = n$. Estes valores substituem os valores de aptidão durante o processo de seleção, e o método procede como o da roleta.

Aplicando-se o método de *ranking* às soluções da Tabela 1, obtém-se um resultado diferente do método da roleta. A Figura 2 compara visualmente as probabilidades geradas por ambos os métodos. Como as probabilidades no método de *ranking* são independentes dos valores de aptidão das soluções, este método pode ser mais interessante quando poucas soluções têm valor de aptidão muito mais alto do que as demais, pois aumenta as chances de soluções de menos qualidade serem selecionadas [33]. Apesar de isso parecer contraintuitivo, é importante permitir que soluções piores contribuam para o desenvolvimento genético da população, pois isso permite uma maior exploração do conjunto de soluções.

Método de torneio: Seja P a população de soluções de um problema. O método de torneio escolhe subconjuntos $S_i \subseteq P$ de soluções da população e então seleciona a melhor solução de cada subconjunto S_i. Assim como o método de *ranking*, este método pode aumentar as probabilidades de seleções piores serem selecionadas, dependendo da maneira como os subconjuntos S_i forem determinados. Geralmente, este método é implementado com torneios de duas soluções, o que o torna uma forma muito rápida de seleção e garante maior variedade de qualidade das soluções escolhidas [31].

Método de Boltzmann: Este método é similar ao RS. Existe um parâmetro

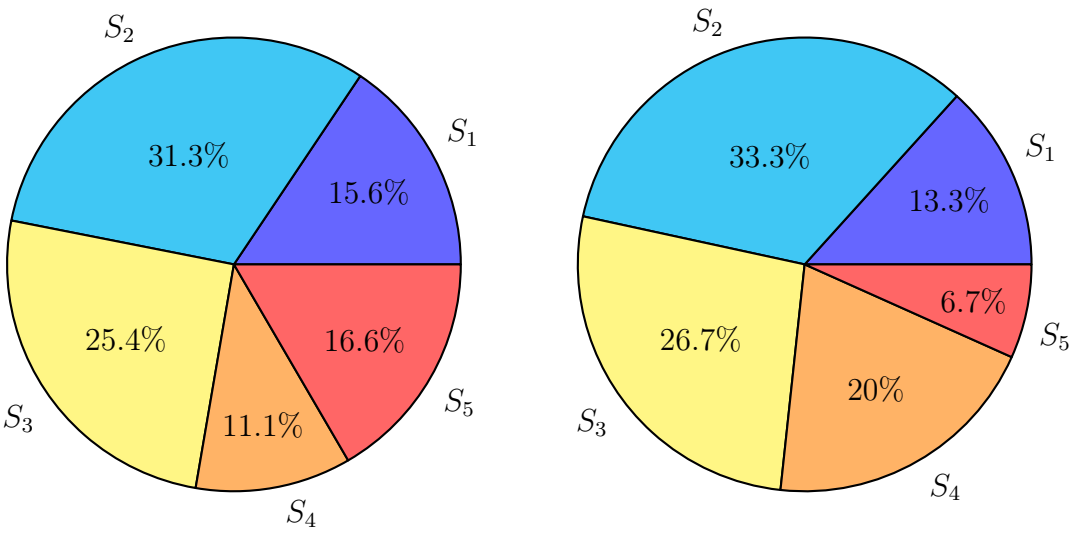

Figura 2: Probabilidades de seleção de soluções pelos métodos da roleta (esquerda) e de *ranking* (direita).

de temperatura que diminui ao longo do tempo. A altas temperaturas, as chances de selecionar quaisquer soluções são quase iguais. Já a temperaturas mais baixas, as chances de selecionar soluções melhores aumentam.

3.1.4 Recombinação

Na etapa de recombinação, também chamada de *crossover*, os cromossomos selecionados são combinados, de modo a gerar novas soluções. Isso pode ser feito de várias maneiras, sendo algumas delas apresentadas a seguir.

Recombinação em k pontos de corte: Este método faz com que cada cromossomo contribua uma ou mais seções para a construção de novas soluções. Para demarcar seções, são definidos k pontos onde um cromossomo do casal começa a contribuir genes e o outro para. Ao alternar a contribuição de cada cromossomo, geram-se dois cromossomos filhos [25]. A Figura 3 ilustra este processo quando $k = 2$. Dois cromossomos de seis genes (à esquerda) são combinados definindo um ponto de troca após o segundo gene e outro após o quarto. Os cromossomos à direita são as duas sequências alternadas possíveis resultantes deste processo.

Recombinação uniforme: Sejam $G_1 = (g_{11}, \ldots, g_{n1})$ e $G_2 = (g_{12,\ldots,g_{n2}})$ cromossomos obtidos na etapa de seleção. A recombinação uniforme passa por cada par de genes (g_{i1}, g_{i2}), com $i = 1, \ldots, n$, e escolhe um dos genes do par para constituir o cromossomo filho. Para tal, é definido um valor de probabilidade de um ou outro gene ser escolhido –

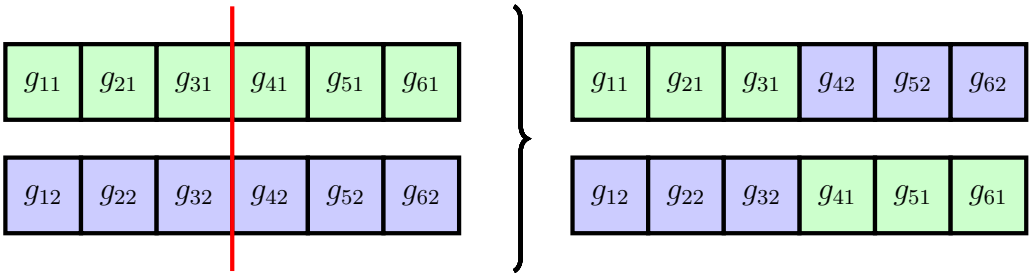

Figura 3: Recombinação em 2 pontos de corte gerando dois novos cromossomos.

geralmente, $p = 0.5$, de modo que as escolhas sejam completamente aleatórias [29]. Computacionalmente, isso costuma ser implementado definindo-se um vetor binário chamado *máscara*, de mesmo tamanho que os cromossomos, onde os números 1 e 0 correspondem a selecionar os genes de um ou outro cromossomo pai. A Figura 4 ilustra o uso de máscara para gerar cromossomos filhos. A alternância no significado dos valores 0 e 1 permite a geração de dois filhos. De certa maneira, a recombinação uniforme pode ser pensada como um método de pontos de corte levado ao extremo, havendo pontos de corte em todos os genes dos cromossomos pais, com a alternância sendo definida pela probabilidade p [5].

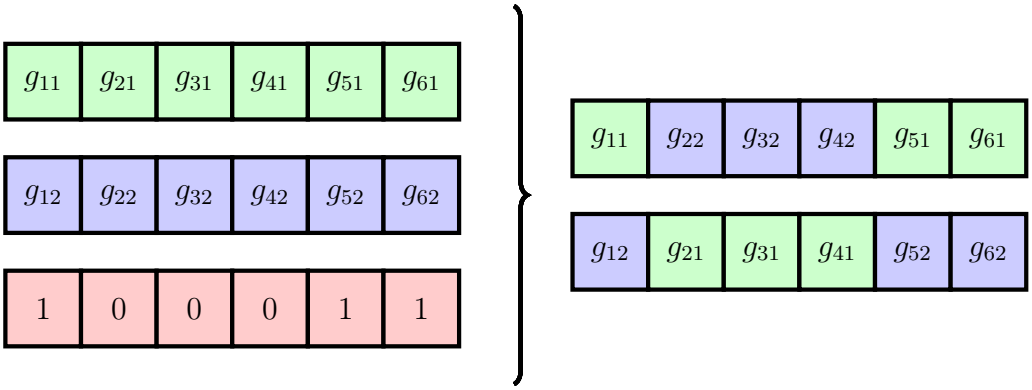

Figura 4: Recombinação uniforme de cromossomos usando máscara.

De acordo com Katoch, Chauhan e Kumar [20], as recombinações em k pontos de corte são preferíveis pela simplicidade de implementação, ao passo que a recombinação uniforme é preferível pela sua maior aleatoriedade e seu potencial para gerar soluções diferenciadas. No entanto, estas técnicas são mais vantajosas quando aplicadas a populações grandes – para populações pequenas, elas podem levar a soluções com pouca diversidade.

Recombinação de mapeamento parcial: Em inglês, *partially matched crossover* (PMX). Sejam $G_1 = (g_{11}, \ldots, g_{n1})$ e $G_2 = (g_{12}, \ldots, g_{n2})$ cromossomos pais. Dois pontos de corte, i e j, com $i < j \leq n$, são definidos, e os genes entre os pontos de corte passam a formar pares (g_{k1}, g_{k2}). Então, para $k = i, \ldots, j$, procura-se pelo gene de G_1 equivalente

a g_{k2} e troca-se a sua posição em G_1 com a posição de g_{k1}. O processo contrário é feito para G_2. A Figura 5 ilustra este processo numericamente. Os pontos de corte definem os pares $(3, 5)$ e $(6, 2)$, ou seja, em ambos os cromossomos, as posições dos genes 3 e 5 são trocadas, bem como as posições dos genes 6 e 2.

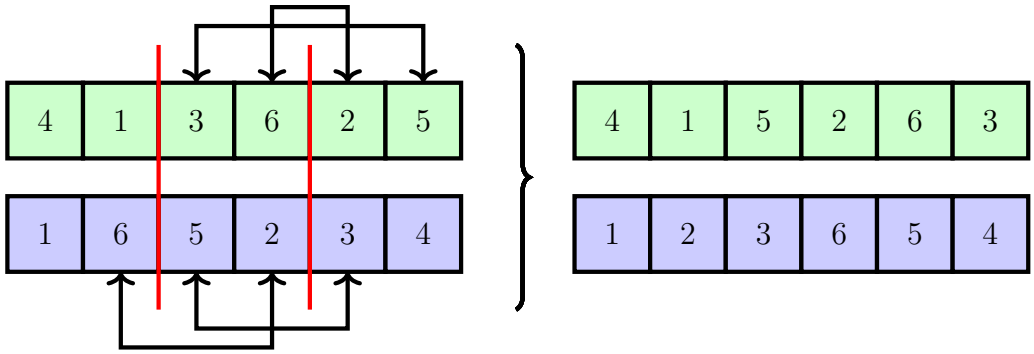

Figura 5: Recombinação de mapeamento parcial, ou PMX.

Recombinações baseadas em ordem: Para determinados problemas, como o problema do caixeiro viajante [28], é desejável manter a ordem de certos cromossomos intacta. Nestes casos, utilizam-se recombinações baseadas em ordem. Segundo Sastry, Goldberg e Kendall [29], o método de Davis [13] é um exemplo deste tipo de recombinação, que começa com a definição de dois pontos de corte entre os cromossomos pais. O conteúdo genético de cada pai entre os pontos de corte é copiado para os cromossomos filhos nas mesmas posições. A Figura 6 ilustra este processo.

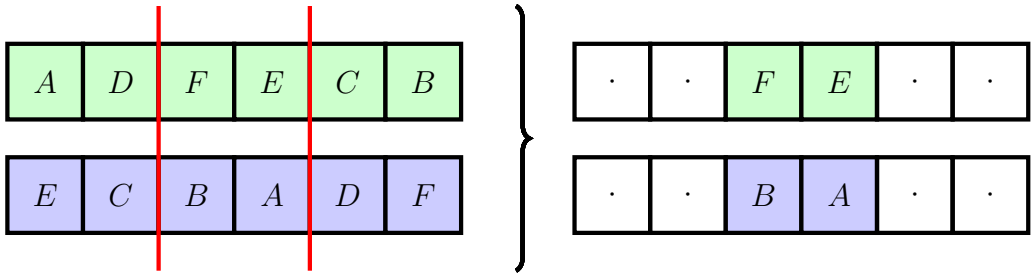

Figura 6: Primeiro passo de uma recombinação baseada em ordem.

Feito este procedimento, procede-se da seguinte maneira: Para o cromossomo filho 1, é feita uma varredura dos genes do cromossomo pai 2. Sempre que um gene diferente dos presentes no cromossomo filho 1 for encontrado, ele é colocado na posição vazia mais próxima do começo do cromossomo filho. Para o cromossomo filho 2, o mesmo processo é feito com o cromossomo pai 1. A Figura 7 mostra o resultado deste processo para os cromossomos da Figura 6.

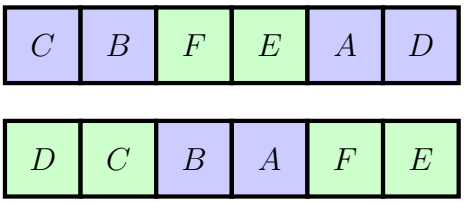

Figura 7: Segundo passo de uma recombinação baseada em ordem.

4 Aplicações de algoritmos genéticos

Como visto até então, um AG é uma ferramenta poderosa para otimização e solução de problemas complexos. Seu uso, embora desafiador, traz novas perspectivas no âmbito da história da invenção. Isto porque os programas de computador passam a poder "evoluir" de forma autônoma, criando projetos que, muitas vezes, um ser humano não poderia realizar. Como visto no projeto do pesquisador John Koza, da Universidade de Stanford nos Estados Unidos, que através de um AG projeta antenas de alto desempenho. Segudo John, nem sempre é claro por que a invenção evoluída funciona: nenhum ser humano poderia, de forma razoável, criar tais antenas, com suas formas estranhas [4].

Essa nova forma de inventar já esta transformando áreas como a robótica, machine learning e as redes neurais.

4.1 Algoritmos genéticos para otimização de redes neurais

O engenheiro de pesquisa Adrian de Wynter propos um problema de escolha de uma arquitetura para uma rede neural como um exercício de aproximação de funções. Nesta formulação, a função é o mapeamento "verdadeiro" dos dados de entrada para as saídas e a aproximação é o modelo de rede neural treinado. As arquiteturas de rede geralmente são escolhidas com base na intuição ou tentativa e erro, mas Wynter afirma que "é improvável que a seleção arbitrária de uma arquitetura neural forneça a melhor solução". Em vez disso, uma pesquisa otimizada de arquitetura automatizada encontraria a combinação desses componentes que aproxima a função com um erro mínimo. O trabalho do Wynter fornece as "garantias teóricas sobre a precisão dos cálculos". Ele mostrou que o problema geral de busca de arquitetura (ASP) é intratável, ou seja, não é possível garantir a execução em tempo polinomial. Assim, propõe uma formulação que se aproxima de uma ASP, que pode ser resolvida em tempo polinomial usando algoritmos genéticos co-evolutivos [3].

4.2 Algoritmos genéticos em problemas de robótica

Em trabalho realizado pela Universidade Federal de São Paulo, foi constatado o comportamento auto-organizável de algoritmos genéticos em problemas associados a robôs móveis e nos quais a função de aptidão se altera. Para lidar com tais problemas não-estacionários, o uso de AGs com Imigrantes Aleatórios, em que o indivíduo com a menor aptidão e seus vizinhos próximos são substituídos em cada geração é proposto. Para que melhores indivíduos da população não levem os novos indivíduos à extinção, estes são preservados em uma subpopulação. A estratégia de substituição apresentada pode levar o

sistema a um comportamento auto-organizável, permitindo que o nível de diversidade da população aumente e que os indivíduos consigam escapar de ótimos locais induzidos pelas mudanças no robô ou no ambiente. A análise dos dados obtidos em simulações sugere que o AG investigado apresenta um tipo de comportamento auto-organizável conhecido como Criticalidade Auto-Organizada, o qual aparece em diversos fenômenos naturais.

Os resultados encontrados constatam que o AG proposto é interessante em problemas em que a nova solução a ser encontrada está localizada em pontos que são dificilmente alcançados, pelos operadores tradicionais, a partir do ótimo local que representa a solução antiga. Assim o AG proposto é interessante quando ocorrem mudanças bruscas no robô ou no ambiente, como falhas que afetaram grande parte dos sensores de distância do robô [12].

Ainda, em pesquisa desenvolvida na Universidade Federal de Ouro Preto foi investigado um algoritmo genético para a solução da cinemática inversa de um manipulador antropomórfico de seis graus de liberdade. A resolução da cinemática inversa é fundamental para transformar as especificações de movimento, atribuídas ao efetuador no espaço operacional, nos correspondentes movimentos espaciais conjuntos, que permitem a execução do movimento desejado. Pórem sua resolução não é trivial e quanto maior o número de graus de liberdade, mais complexa se torna a solução. A análise geométrica da posição do efetuador para encontrar os valores angulares das juntas se traduz em um conjunto extenso de equações não lineares. O problema analisado dessa forma pode ter múltiplas soluções ou até mesmo não possuir soluções admissíveis. Considerando que existem múltiplas soluções viáveis dentro do espaço de trabalho deste tipo de robô, a principal motivação é, dada uma pose de referência do efetuador, encontrar a melhor configuração das juntas que implicará em um ganho de eficiência energética e tempo de operação. O objetivo do algoritmo desenvolvido na pesquisa é minimizar o deslocamento angular com margens de erro de posicionamento e orientação aceitáveis. Para identificar a melhor solução são analisados três fatores fundamentais: o tempo de execução do algoritmo até atingir o valor ótimo, o erro de médio de posição e o erro máximo dos ângulos de orientação, conforme equações de distância euclidiana. Os resultados da pesquisa constataram que o principal benefício do método apresentado é a facilidade de adaptação da solução, o que permite aplicá-la a diferentes processos robotizados, bastando adaptar a cinemática do robô e seus limites de juntas [23].

5 Um problema prático

Para testar as características de AGs discutidas anteriormente, optou-se por implementar o AG descrito por Sedighi et al. [30]. Trata-se de um método para determinar a rota que um robô móvel deve seguir em um mapa de modo a concluir um trajeto que minimize a distância percorrida, o número de curvas e o número de colisões contra obstáculos. Para implementar este AG e testar seus resultados, utilizou-se a linguagem de programação Julia [8]. As seções seguintes são dedicadas a explicar as etapas do AG implementado, bem como as escolhas feitas durante a implementação do código, que pode ser encontrado em repositório virtual.

5.1 Componentes do mapa

Assume-se que o robô tenha o mapa completo do ambiente, visto de cima, antes de iniciar o AG, e que os obstáculos sejam retangulares. Para que a roteirização possa ser implementada, é necessário discretizar o mapa, de modo que existam pontos bem-definidos pelos quais o robô possa passar. De fato, o próprio robô é considerado pontual neste sistema.

Para definir um mapa, inicia-se definindo os obstáculos presentes. Isso é feito por meio da função `obstaculo`, que recebe as coordenadas e as dimensões do objeto no plano. Após criar uma lista de obstáculos, é necessário fazer a discretização do mapa por meio da função `criar_ecra`. Esta função recebe a lista de obstáculos e a largura do mapa – como é explicado adiante, este AG depende de mapas quadrados para funcionar; portanto, caso um mapa não seja quadrado, é necessário distorcê-lo de acordo com o necessário. Um parâmetro adicional de *tamanho de passo*, Δ, pode ser especificado. Trata-se da distância entre dois pontos da discretização, que, por padrão, é 1. O que a função `criar_ecra` retorna é uma matriz *booleana* quadrada especificando quais pontos discretizados correspondem a espaços livres no mapa, e quais correspondem a espaços ocupados por obstáculos.

Para visualizar o mapa criado, basta chamar a função `mapear`, que recebe a lista de obstáculos e a largura do mapa como argumentos. Opcionalmente, é possível passar a matriz de ecrã para esta função, junto com o tamanho de passo, para mostrar o resultado da discretização do mapa. A Figura 8 mostra o exemplo de um mapa e sua discretização. As partes brancas do mapa são locais pelos quais o robô pode passar; as partes pretas são obstáculos. Os pontos representam a discretização do mapa, sendo azuis os pontos pelos quais o robô pode passar e vermelhos os pontos indicativos de colisão. O Código 9 foi usado para gerar este mapa.

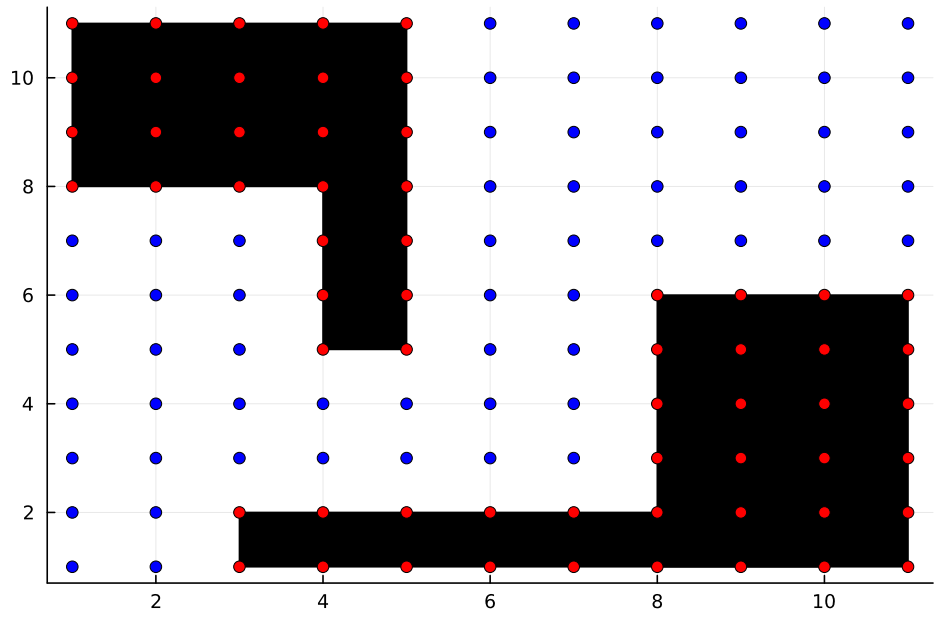

Figura 8: Exemplo de mapa discretizado.

```
obs = [obstaculo(1, 4, 8, 3),
obstaculo(4, 1, 5, 3),
obstaculo(3, 7, 1, 1),
obstaculo(8, 3, 1, 5)]
largura = 10
Δ = 1
E = criar_ecra(obs, largura, Δ=Δ)
grafico = mapear(obs, largura, ecra=E, Δ=Δ)
```

Código 9: Código para geração de mapa.

5.2 Estrutura cromossomial

Os cromossomos propostos por Sedighi et al. [30] são divididos em quatro segmentos, como ilustra a Figura 10. O primeiro alelo contém um gene binário chamado *path* (caminho). O papel deste gene é indicar se o robô realizará deslocamentos *row-wise* ou *column-wise*. Estes movimentos só podem ser entendidos com a descrição da seção *location* (localização), que consiste em n alelos que descrevem uma sequência de pontos discretizados pelos quais o robô deve passar.

Quando o gene em *path* indica movimento *row-wise*, o robô começa na linha 1, e cada deslocamento incrementa em 1 o número da linha na qual o robô se encontra, ao passo que o número da coluna é definido pelo respectivo gene na seção *location*. Assim,

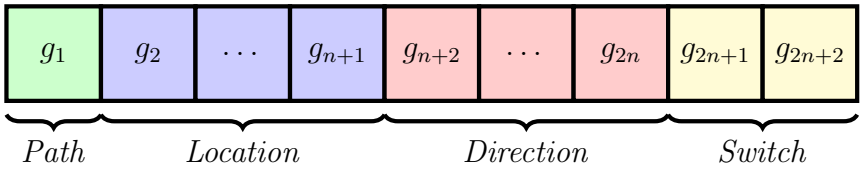

Figura 10: Cromossomo para roteirização [30].

se *location* for o vetor $(x_1, \ldots, x_n)$, a sequência de deslocamentos *row-wise* do robô será $\big((1, x_1), (2, x_2), \ldots, (n, x_n)\big)$. Em outras palavras, no deslocamento *row-wise*, o robô se desloca independentemente de uma linha para a outra, e os genes em *location* especificam as colunas para as quais o robô deve se mover.

Se *path* indicar deslocamento *column-wise*, o oposto acontece; ou seja, o deslocamento do robô em termos das colunas é independente, e a seção *location* descreve as linhas para as quais o robô deve se mover. Sequencialmente, isso equivale a $\big((x_1, 1), (x_2, 2), \ldots, (x_n, n)\big)$.

Na implementação deste projeto, considerou-se que o valor 0 para *path* indica movimento *row-wise* e 1 indica movimento *column-wise*. Ademais, cabe observar que os genes em *location* podem conter quaisquer valores inteiros entre 1 e n, sendo que $(1, 1)$ é o ponto discretizado de origem do mapa, e n é o número de linhas existentes na discretização. Como os movimentos *row-wise* e *column-wise* asseguram que todo deslocamento faça o robô se mover para um ponto diferente do atual, é aceitável a existência de genes repetidos em *location*. As Figuras 11 e 12 ilustram os trajetos gerados por deslocamentos *row-wise* e *column-wise* para um cromossomo com *location* $(1, 3, 1, 2, 2, 6)$. Neste caso, os trajetos do robô são representados, respectivamente, pelas sequências $\big((1, 1), (2, 3), (3, 1), (4, 2), (5, 2), (6, 6)\big)$ e $\big((1, 1), (3, 2), (1, 3), (2, 4), (2, 5), (6, 6)\big)$.

As Figuras 11 e 12 assumem, implicitamente, que o movimento do robô segue um padrão de geometria táxi (ou Manhattan), em que o robô decompõe o percurso de menor distância até a próxima coordenada em trajetos puramente verticais e horizontais [11]. Os movimentos são assim definidos porque o deslocamento euclidiano equivalente pode resultar em colisões inesperadas com as quinas dos obstáculos retangulares. No entanto, existe uma questão a resolver: o robô deve fazer o movimento horizontal antes do vertical, ou o contrário? É para responder esta pergunta que a seção *direction* (direção) existe no cromossomo. *Direction* consiste em um vetor de $n - 1$ genes binários cujo k-ésimo gene descreve a ordem dos movimentos realizados do k-ésimo ponto de *location* ao seu $(k + 1)$-ésimo ponto. Nesta implementação, considera-se que o gene 0 indica que o movimento horizontal antecede o vertical, e o gene 1 indica o contrário. Ordens de movimentos

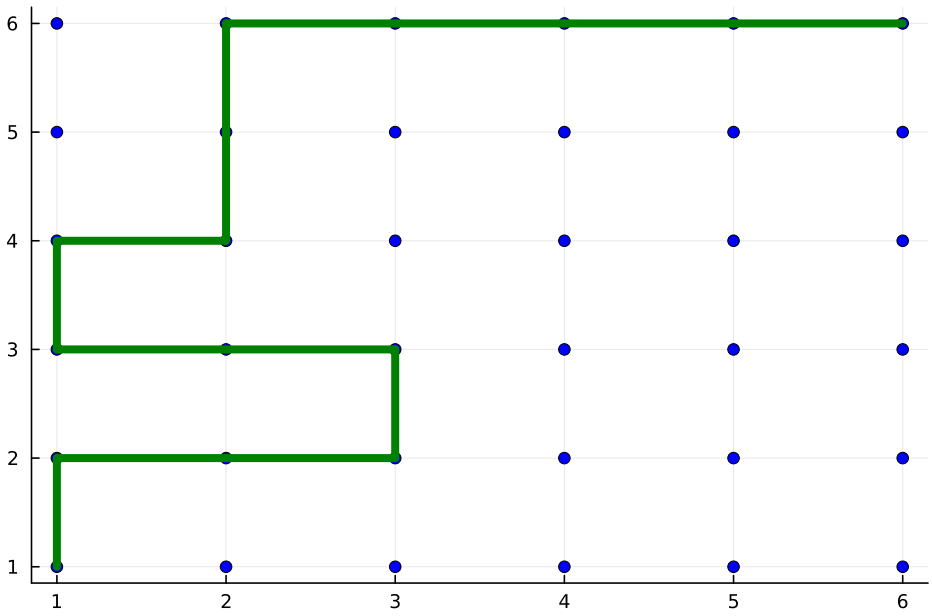

Figura 11: Deslocamento *row-wise*.

inadequadas podem levar a colisões; por isso, o AG deve convergir para uma solução com uma sequência *direction* adequada.

De acordo com Sedighi et al. [30], as três primeiras seções do cromossomo podem gerar resultados satisfatórios para problemas de menor complexidade [15, 18]. Contudo, alguns mapas podem ter curvas muito complexas, o que pode exigir maior versatilidade dos movimentos do robô. Para contornar este problema, introduz-se a quarta seção do cromossomo, *switch*. Sejam $1 \leq s_1 \leq n$ e $1 \leq s_2 \leq n$ os genes da seção *switch*. Se o alelo atual da seção *location* estiver na posição s_1 ou s_2, o movimento do robô é alterado de *row-wise* para *column-wise*, e vice-versa. Na ocasião em que $s_1 = s_2$, a troca de movimento não ocorre. Do contrário, é possível que o robô alterne até duas vezes o seu tipo de movimento.

5.3 Função de aptidão

Para mensurar a aptidão de cada cromossomo em uma população, Sedighi et al. [30] desenvolveram um método que requer três vetores: `len`, `col` e `turns`. O vetor `len` contém a distância total percorrida pelo robô no caminho definido por cada cromossomo; `col` contém o número de colisões do robô com cada obstáculo; `turns` contém o número de vezes que o robô precisa executar uma rotação de 90°. Obtidos estes vetores, seus valores são mapeados para o intervalo $[0, 1]$. Por exemplo, seja u um dos vetores a mapear. Então

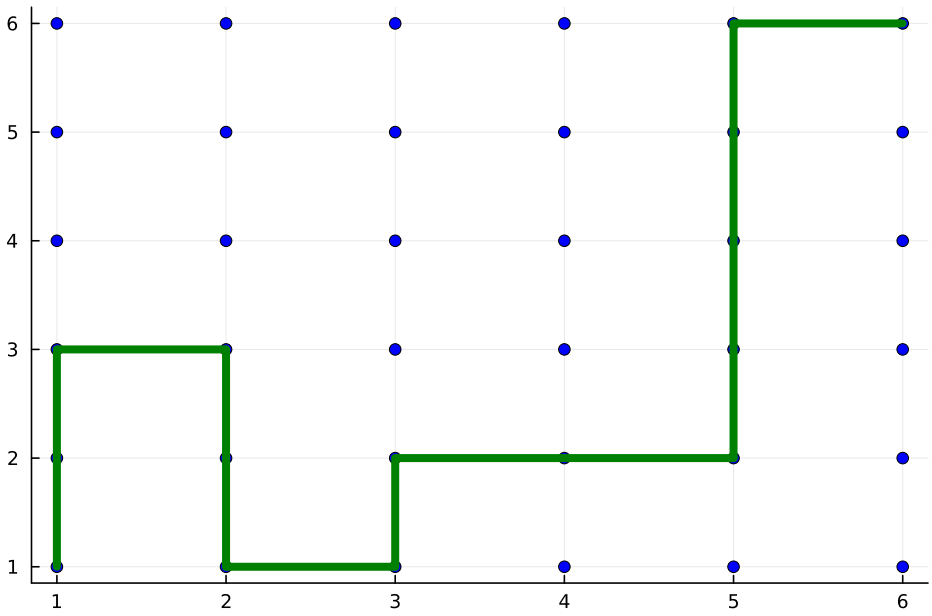

Figura 12: Deslocamento *column-wise*.

cada componente do vetor v resultante do mapeamento é dada por $v_k = C(u_k - \min(u)) + 1$, sendo C o coeficiente

$$C \coloneqq -\frac{1}{\max(u) - \min(u)}.$$

Ou seja, os valores de `len` são mapeados de 1 a 0, da menor distância percorrida até a maior; os valores de `col` são mapeados do menor ao maior número de colisões; e os valores de `turns` do menor ao maior número de rotações.

Duas funções de aptidão são definidas. Sejam ℓ, c e t as versões mapeadas dos vetores `len`, `col` e `turns`. A primeira função de aptidão, ϕ, é aplicada a cromossomos que não resultam em colisões contra obstáculos:

$$\phi = c_k(L\ell_k + Tt_k)\frac{100}{L+T}.$$

Nesta função, L e T são, respectivamente, recompensas por minimização da distância percorrida e rotações realizadas. Sedighi et al. [30] sugerem definir $L \coloneqq 1$ e $T \coloneqq 2$. A justificativa é que as rotações de 90° tendem a ser mais demoradas do que a translação do robô. Alternativamente, se o cromossomo levar o robô a colidir com obstáculos, é utilizada a função ψ:

$$\psi = 0.1\frac{\phi_k}{\texttt{col}_k},$$

sendo ϕ_k o retorno da função ϕ e $\texttt{col}_k$ o número de colisões do cromossomo k. A consequência desta definição é que cromossomos que levam a colisões são fortemente penalizados.

5.4 Manutenção da população

Determinada a aptidão de cada cromossomo da população, é feita seleção por *ranking*. Os cromossomos pais são recombinados utilizando 1 ponto de corte, gerado aleatoriamente em uma das três primeiras seções do cromossomo. Quanto à mutação, Sedighi et al. [30] sugerem definir uma probabilidade de mutação em cada gene. Em seguida, gera-se um número aleatório que, se estiver dentro desta probabilidade, modifica o valor do gene, respeitando os conjuntos aceitáveis de valores de cada gene. Por fim, os autores sugerem a utilização da técnica elitista que envolve copiar o melhor cromossomo da população atual para a próxima.

5.5 Testes computacionais

Nesta seção, são abordados alguns testes computacionais realizados com o algoritmo genético descrito. Cabe explicar que, para simular uma população de soluções para determinado problema, o usuário deve criar um mapa e em seguida chamar a função `algoritmo_genetico`. Além disso, por propósitos de consistência e comparação, todos os cromossomos foram definidos de tal forma que o trajeto comece na coordenada $(1, 1)$ e termine na coordenada $(11, 11)$, e todos os mapas testados são 10×10, com tamanho de passo 1.

5.5.1 Mapa A

O primeiro teste foi conduzido com o mapa da Figura 8. A princípio, considerou-se uma população de 10 cromossomos, ao longo de 100 gerações, com probabilidade de mutação de 2.5%. A Figura 13 mostra o trajeto do melhor cromossomo obtido ao fim do processo. A Figura 14 mostra a aptidão da melhor resposta, da média e da pior resposta de cada geração deste experimento.

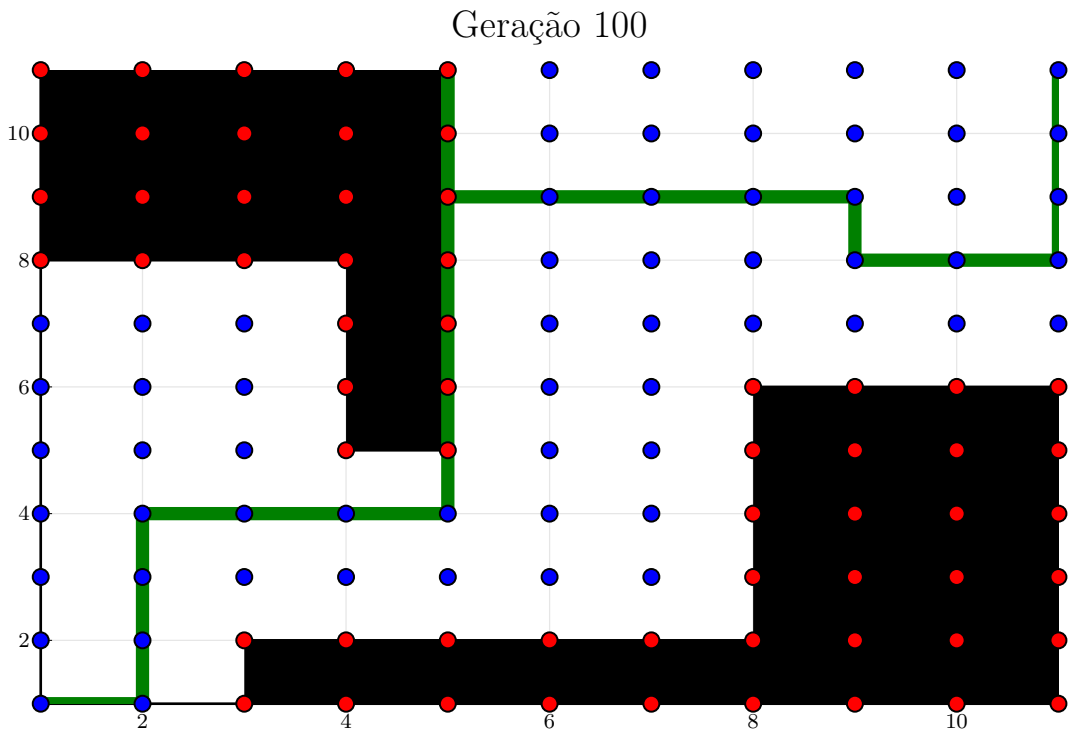

Figura 13

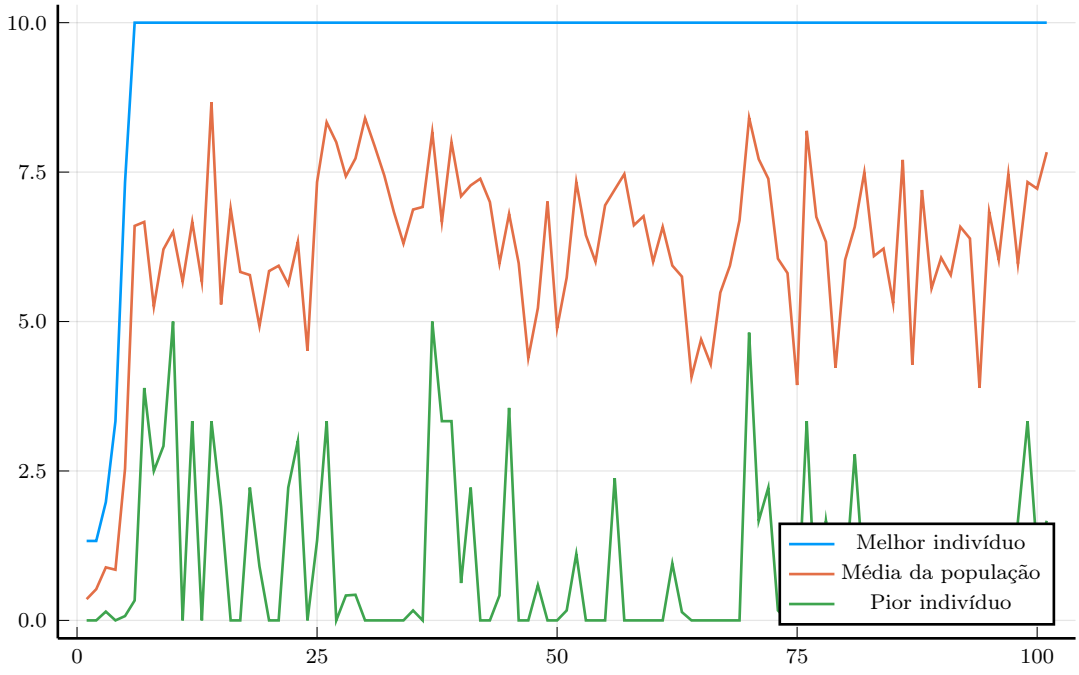

Figura 14

Observa-se que os valores de aptidão de todos os cromossomos da população estão relativamente próximos. Convém lembrar que as funções de aptidão deste método são definidas de modo que exista uma grande diferença entre soluções sem colisão e soluções com colisão. Portanto, com alguns ajustes de parâmetros, espera-se observar uma mudança abrupta nos valores de aptidão. Supondo que os resultados ruins deste experimento se devam à população pequena, aumentou-se o número de cromossomos de cada população para 100, resultando nas Figuras 15 e 16.

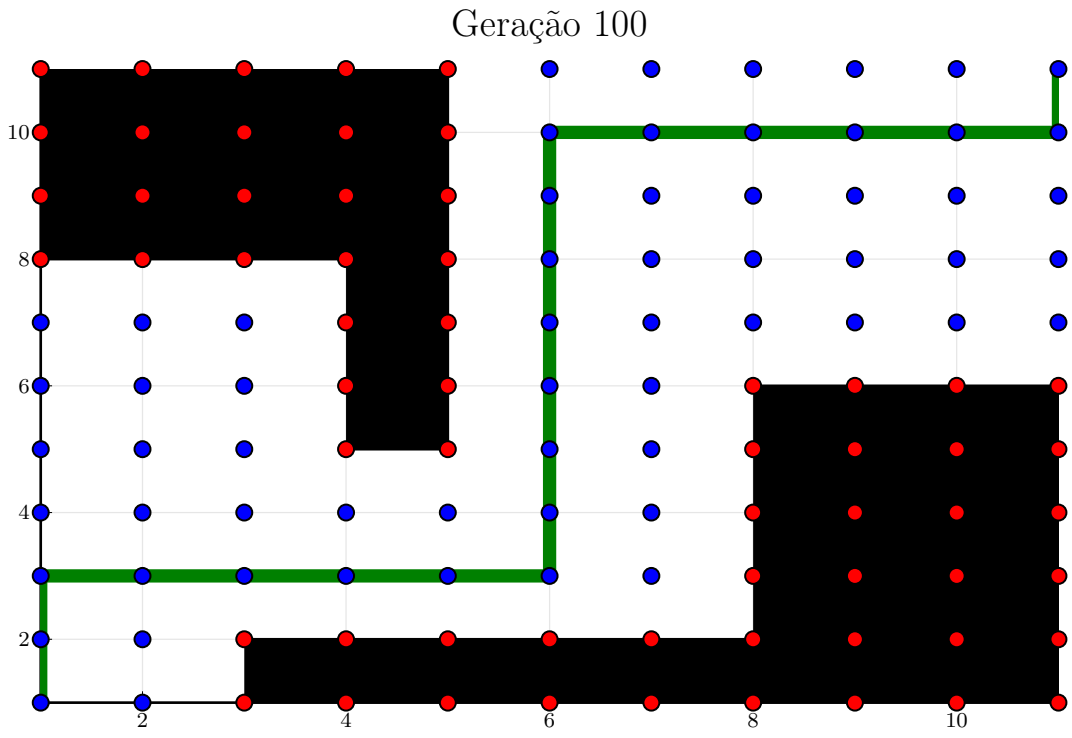

Figura 15

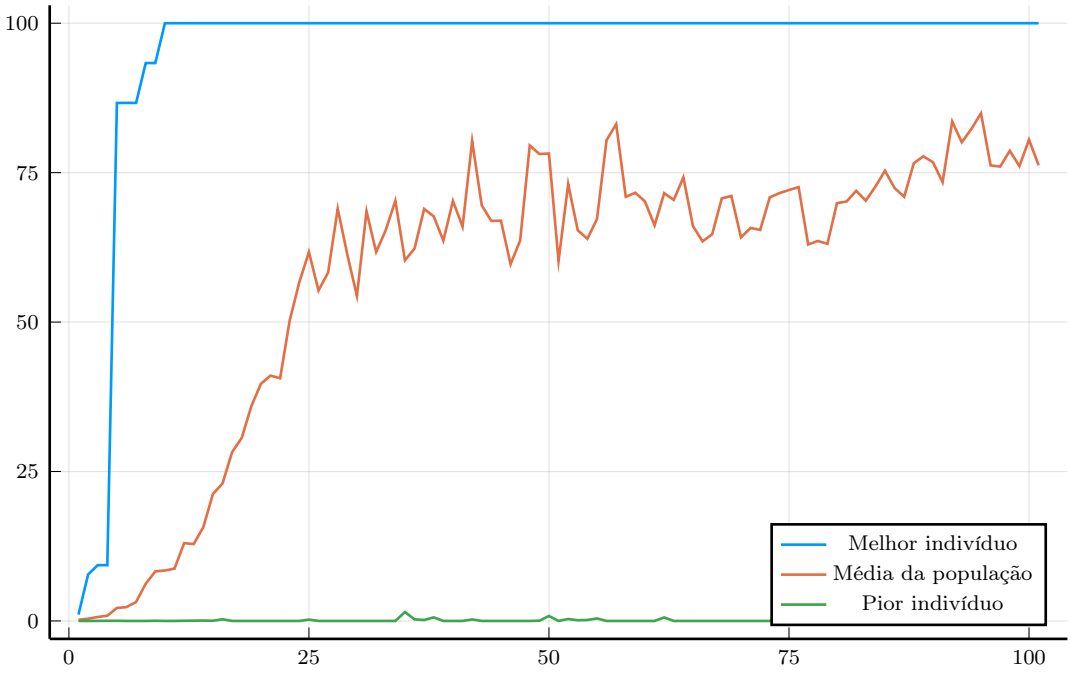

Figura 16

Como no caso anterior, observa-se uma tendência do melhor indivíduo a chegar a um valor aproximado de 10; porém, desta vez, ocorreu uma explosão no valor de aptidão do melhor indivíduo, levando-o a aptidão 100 em menos de 20 gerações. A média da população se aproxima do valor do melhor indivíduo mais lentamente, e o pior indivíduo de cada população se mantém numa faixa de valores muito restrita. Neste caso, parece que aumentar o tamanho da população foi a escolha correta: presume-se que, com isso, a variedade genética inicial foi suficiente para que combinações cromossomiais de maior qualidade surgissem cedo.

5.5.2 Mapa B

O segundo mapa testado contém um obstáculo central. Com uma população de 100 cromossomos, foram simuladas 100 gerações com chance de mutação de 2.5% por gene. O resultado final é apresentado nas Figuras 17 e 18.

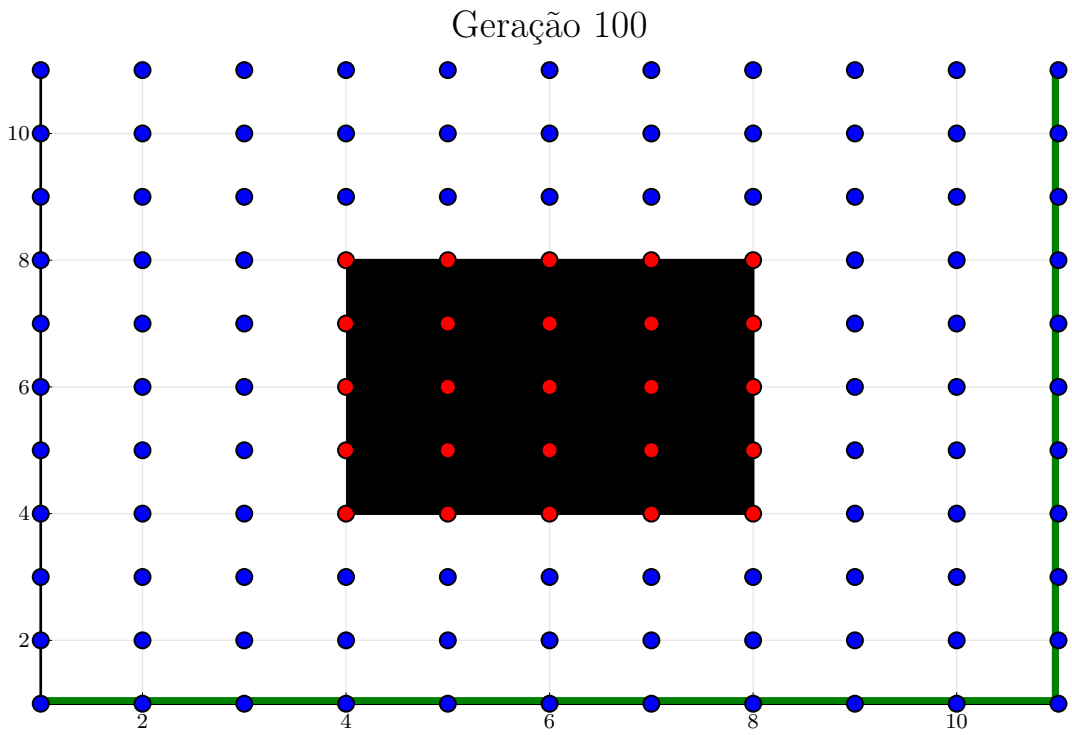

Figura 17

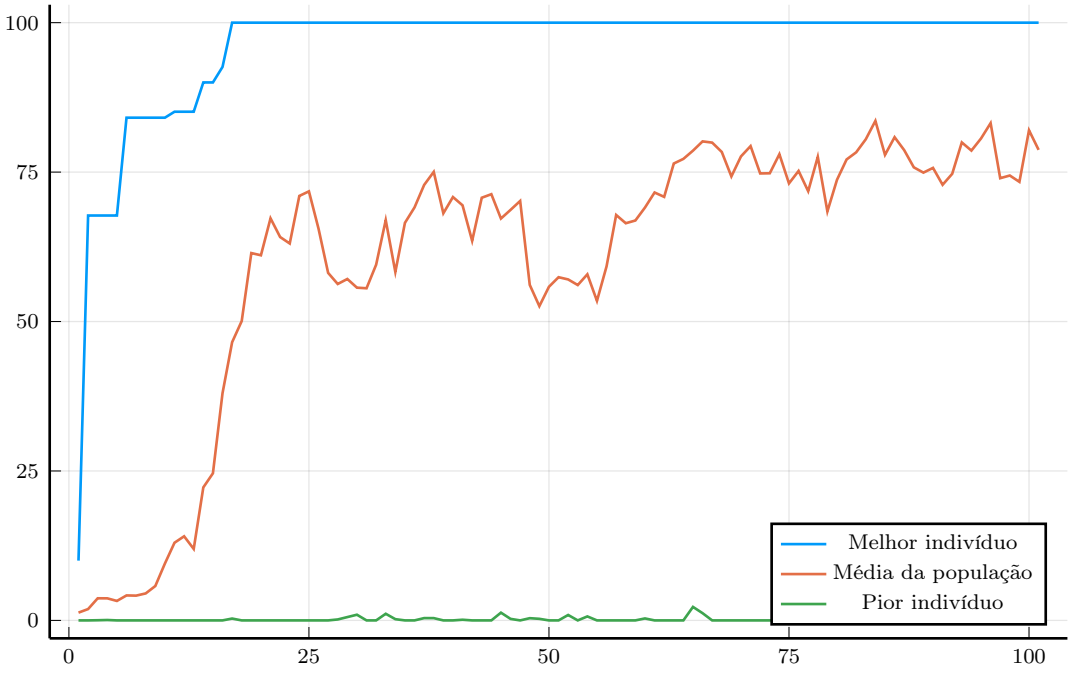

Figura 18

O gráfico de aptidões se assemelha ao do mapa A. É interessante comparar o resultado deste experimento ao de um mapa sem obstáculos. Testando o mapa vazio com 100 cromossomos por população e 1000 gerações de simulação, obtêm-se os resultados das Figuras 19 e 20. Neste teste, o trajeto final do robô acaba envolvendo três rotações de 90°, ao passo que no teste com obstáculo, apenas duas rotações são utilizadas. É fácil ver que o trajeto desenhado com obstáculo também é o ótimo para o caso sem obstáculo, o que indica que, na ausência de obstáculos, o algoritmo apresenta uma tendência a escolher rotas sub-ótimas. Isso pode ser explicado pelo fato de a diferenciação entre soluções que usam a mesma função de aptidão não ser notável o suficiente para que as soluções se direcionem em direção ao ótimo global do problema.

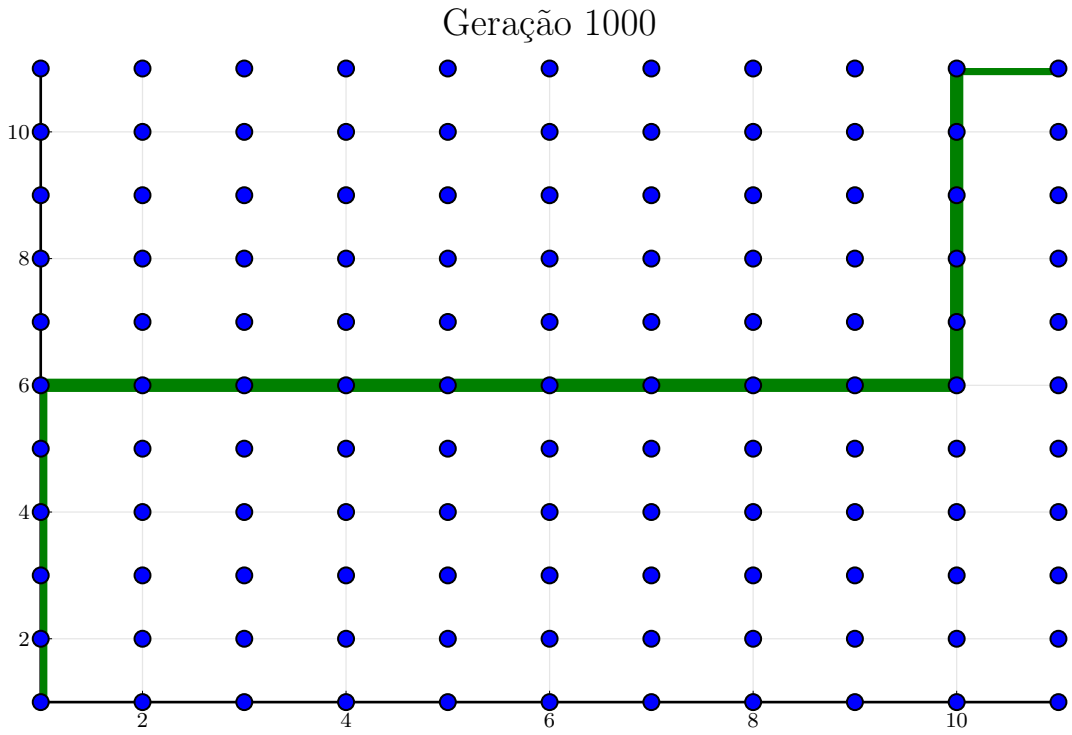

Figura 19

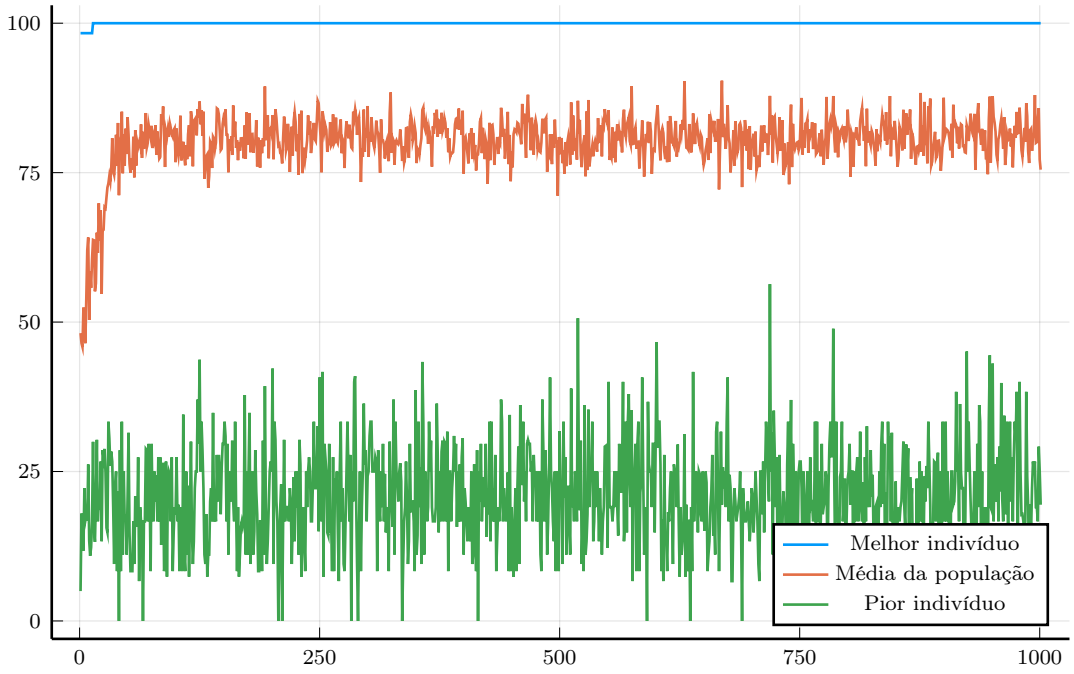

Figura 20

5.5.3 Mapa C

Este mapa consiste de um ecrã 14×14, com 3 obstáculos. O mapa apresenta dois gargalos que podem ser difíceis de superar. Após algumas simulações sem sucesso com 100 cromossomos e 1000 gerações, a chance de mutação por gene foi aumentada de 2.5% para 5%, e, desta vez, obtiveram-se os resultados das Figuras 21 e 22. O fato de a média populacional se manter baixa ao longo das 1000 gerações é um indicativo do nível de dificuldade deste problema. Ademais, é importante observar a característica quase constante do gráfico do melhor indivíduo. Este comportamento acontece devido ao elitismo que preserva o melhor indivíduo de cada geração. Assim, tem-se a garantia de que a melhor solução tem aptidão monótono não-decrescente, o que ajuda muito na resolução de problemas complexos como este.

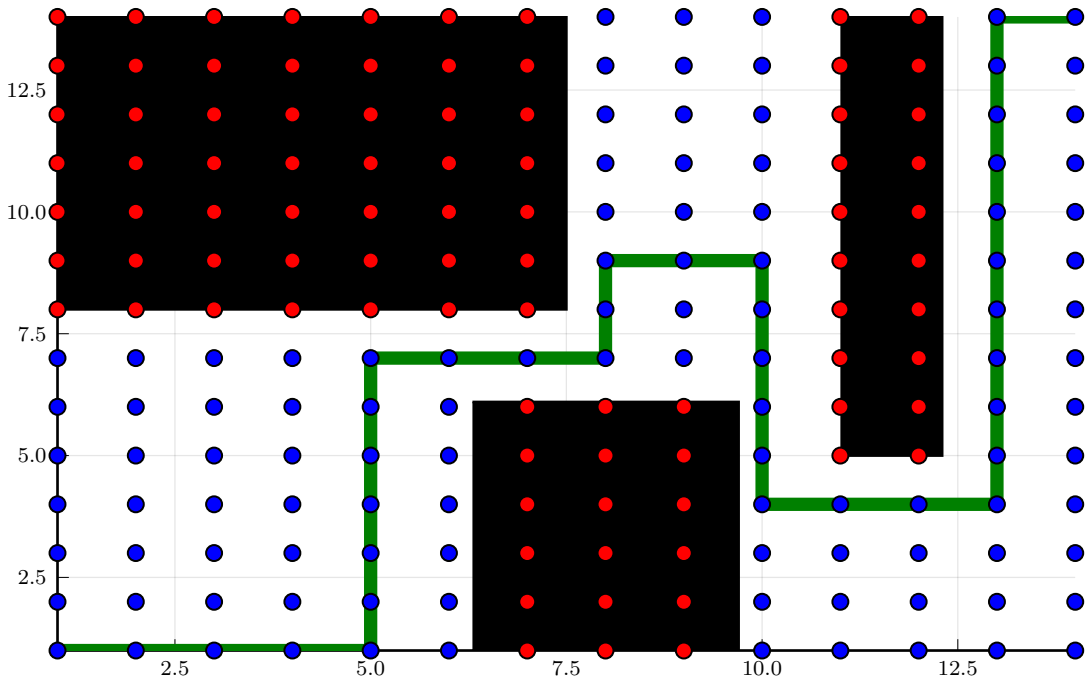

Figura 21

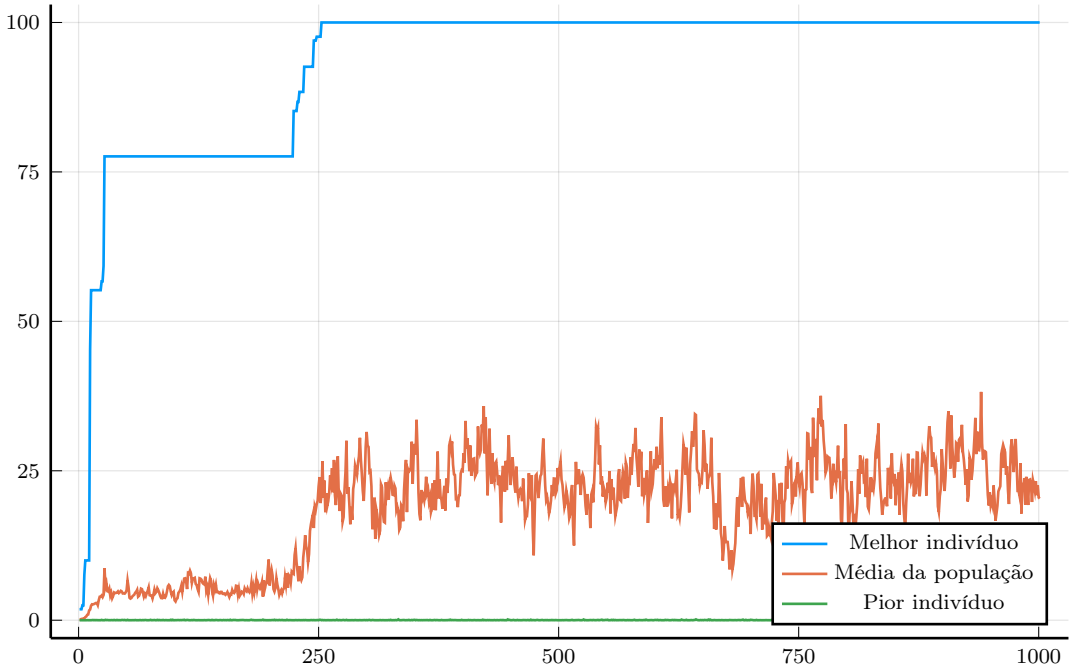

Figura 22

Referências

[1] *A saga das mariposas*. URL: https://cienciahoje.org.br/artigo/a-saga-da-mariposa/ (acesso em 06/07/2016).

[2] Stéphane Alarie, Charles Audet, Aïmen E. Gheribi, Michael Kokkolaras e Sébastien Le Digabel. "Two Decades of Blackbox Optimization Applications". Em: *EURO Journal on Computational Optimization* 9 (2021), p. 100011. DOI: 10.1016/j.ejco.2021.100011. (Acesso em 05/06/2024).

[3] *Algoritmos genéticos oferecem melhor solução para otimização de redes neurais*. URL: https://www.infoq.com/br/news/2020/02/alexa-genetic-deep-learning/ (acesso em 25/02/2020).

[4] *Algoritmos genéticos: As invenções que evoluem*. URL: https://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=invencoes-evolutivas-algoritmos-geneticos&id=010150110526 (acesso em 26/05/2011).

[5] João Carlos Holland De Barcellos. "Algoritmos genéticos adaptativos: um estudo comparativo." Mestrado em Sistemas Digitais. São Paulo: Universidade de São Paulo, 2000. DOI: 10.11606/D.3.2000.tde-05092001-141334. (Acesso em 10/06/2024).

[6] Mokhtar S. Bazaraa, John J. Jarvis e Hanif D. Sherali. *Linear Programming and Network Flows*. 4ª ed. Wiley, 2010.

[7] Dimitris Bertsimas e John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Dynamic Ideas, 1997.

[8] Jeff Bezanson, Alan Edelman, Stefan Karpinski e Viral B. Shah. *Julia: A Fresh Approach to Numerical Computing*. 2015. DOI: 10.48550/arXiv.1411.1607. arXiv: 1411.1607 [cs]. (Acesso em 11/06/2024).

[9] André Ponce de Leon F. de Carvalho. *Algoritmos Genéticos*. https://sites.icmc.usp.br/andre/research/genetic/. (Acesso em 05/06/2024).

[10] Pedro Henrique Centenaro. *Fundamentos Matemáticos e Computacionais de Problemas de Roteamento de Veículos*. 2023.

[11] Sulamita Maria Comini César e Dra Eliane Scheid Gazire. "MINICURSO DE GEOMETRIA TÁXI". Em: (2010).

[12] *Comportamento auto-organizável em algoritmos genéticos aplicados a robôs móveis em ambientes dinâmicos*. URL: https://www.scielo.br/j/ca/a/pPYRLxMpynVrCv9SrdDhY6y/?lang=pt (acesso em 18/03/2007).

[13] Lawrence Davis. "Applying Adaptive Algorithms to Epistatic Domains". Em: *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'85. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985, pp. 162–164. (Acesso em 10/06/2024).

[14] Daniel Delahaye, Supatcha Chaimatanan e Marcel Mongeau. "Simulated Annealing: From Basics to Applications". Em: *Handbook of Metaheuristics*. Ed. por Michel Gendreau e Jean-Yves Potvin. Vol. 272. Cham: Springer International Publishing, 2019, pp. 1–35. DOI: 10.1007/978-3-319-91086-4_1. (Acesso em 05/06/2024).

[15] T. Geisler e T.W. Manikas. "Autonomous Robot Navigation System Using a Novel Value Encoded Genetic Algorithm". Em: *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.* Vol. 3. 2002, pp. III–III. DOI: 10.1109/MWSCAS.2002.1186966. (Acesso em 16/06/2024).

[16] Michel Gendreau e Jean-Yves Potvin, ed. *Handbook of Metaheuristics*. Vol. 146. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010. DOI: 10.1007/978-1-4419-1665-5. (Acesso em 03/06/2024).

[17] J.A. George e D.F. Robinson. "A Heuristic for Packing Boxes into a Container". Em: *Computers & Operations Research* 7.3 (1980), pp. 147–156. DOI: 10.1016/0305-0548(80)90001-5. (Acesso em 06/04/2024).

[18] Aditia Hermanu, Kaveh Ashenayi, Theodore W Manikas e Roger L Wainwright. "AUTONOMOUS ROBOT NAVIGATION USING A GENETIC ALGORITHM WITH AN EFFICIENT GENOTYPE STRUCTURE". Em: (2002).

[19] Denny Hermawanto. "Genetic Algorithm for Solving Simple Mathematical Equality Problem". Em: (2013).

[20] Sourabh Katoch, Sumit Singh Chauhan e Vijay Kumar. "A Review on Genetic Algorithm: Past, Present, and Future". Em: *Multimedia Tools and Applications* 80.5 (2021), pp. 8091–8126. DOI: 10.1007/s11042-020-10139-6. (Acesso em 17/04/2024).

[21] Fei Liu, Chengyu Lu, Lin Gui, Qingfu Zhang, Xialiang Tong e Mingxuan Yuan. *Heuristics for Vehicle Routing Problem: A Survey and Recent Advances*. 2023. arXiv: 2303.04147 [cs, math]. (Acesso em 05/06/2024).

[22] Diogo Correa Lucas. *Algoritmos Genéticos: um estudo de seus conceitos fundamentais e aplicação no problema de grade horária*. 2000. URL: https://www2.ufpel.edu.br/prg/sisbi/bibct/acervo/info/2000/Mono-Diogo.pdf.

[23] Gustavo Medeiros Freitas e Gustavo Pessin Matheus Alves e Farnese. "Algoritmos Genéticos para Determinação da Cinemática Inversa de um Robô de 6DoF". Em: *Sociedade Brasileira de Automática* 1.1 (2020).

[24] Carsten Murawski e Peter Bossaerts. "How Humans Solve Complex Problems: The Case of the Knapsack Problem". Em: *Scientific Reports* 6.1 (2016), p. 34851. DOI: 10.1038/srep34851. (Acesso em 05/06/2024).

[25] Johnny Alexander Bastidas Otero. "ALGORITMOS GENÉTICOS APLICADOS À SOLUÇÃO DO PROBLEMA INVERSO BIOMAGNÉTICO". MESTRE EM ENGENHARIA ELÉTRICA. Rio de Janeiro, Brazil: PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, 2016. DOI: 10.17771/PUCRio.acad.28372. (Acesso em 10/06/2024).

[26] Ademir Alves Ribeiro e Elizabeth Wegner Karas. *Otimização Contínua: Aspectos Teóricos e Computacionais*. 1ª ed. Curitiba: Editora Cengage, 2013.

[27] Olympia Roeva, Stefka Fidanova e Marcin Paprzycki. "Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling". Em: (2013).

[28] Amanur Rahman Saiyed. "The Traveling Salesman Problem". Em: (2012).

[29] "Genetic Algorithms". Em: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. por Kumara Sastry, David Goldberg e Graham Kendall. New York: Springer, 2005.

[30] K.H. Sedighi, K. Ashenayi, Theodore Manikas, R.L. Wainwright e Heng-Ming Tai. "Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm". Em: *IEEE Congress on Evolutionary Computation*. Vol. 2. 2004, 1338–1345 Vol.2. DOI: 10.1109/CEC.2004.1331052.

[31] Anupriya Shukla, Hari Mohan Pandey e Deepti Mehrotra. "Comparative Review of Selection Techniques in Genetic Algorithm". Em: *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. 2015, pp. 515–519. DOI: 10.1109/ABLAZE.2015.7154916. (Acesso em 06/06/2024).

[32] James Stewart. *Cálculo*. 8ª ed. Vol. 1. Cengage Learning, 2016.

[33] Saneh Lata Yadav e Asha Sohal. “Comparative Study of Different Selection Techniques in Genetic Algorithm”. Em: *International Journal of Engineering* 6.3 (2017).

[34] Xiaolan Zhang. “Linear Programming”. Em: (2020).