

Algoritmos genéticos aplicados a robótica

Gabriela Custódio Martins

Gustavo Storithont Mudri

Pedro Henrique Centenaro

1 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 [26]:

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, \dots, x_n os pontos críticos de f . Calcular $y_k = f''(x_k) \forall k \in \{1, \dots, n\}$.
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 $x \rightarrow -\infty \Rightarrow f(x) \rightarrow -\infty$ e $x \rightarrow \infty \Rightarrow f(x) \rightarrow -\infty$. 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 [28]. 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

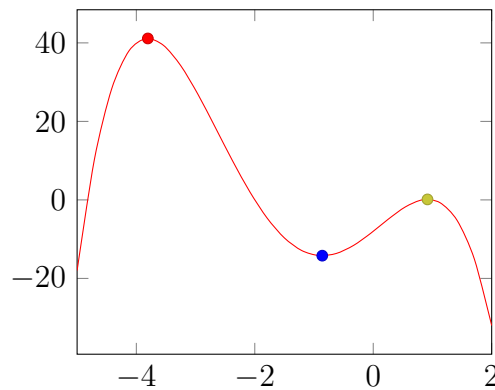


Figura 1: Pontos de máximo e mínimo de uma função

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 [3, 4], 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 [7], 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 [19].

1.1 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 [12] 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* [15], 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 [10], 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.

2 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.

2.1 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 [13], 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. As posições dos dados contidos em um cromossomo são chamadas de alelos, e os valores contidos nestas posições são chamados de genes do cromossomo.

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 [11].

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 [22], 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 [1]. Feito isso, resta implementar os passos seguintes, cuja descrição é baseada, principalmente, na obra de Sastry, Goldberg e Kendall [22].

2.1.1 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 especializadas. 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 [17] 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 cromossomal e, conseqüentemente, 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 [20]. Para problemas com custo computacional muito elevado, Delahaye, Chaimatanan e Mongeau [10] recomendam a utilização de métodos sem população, como o RS.

2.1.2 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 [13] 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.

2.1.3 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 ($i = 1, \dots, n$) 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 $k = 2, \dots, n$, 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 [6, 22].

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 atribuído 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 [27]. 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

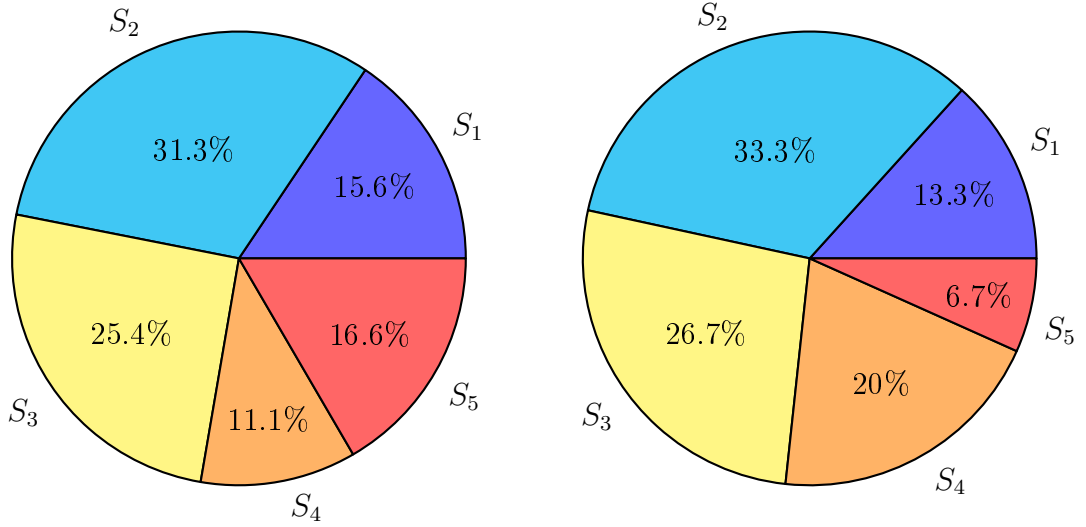


Figura 2: Probabilidades de seleção de soluções pelos métodos da roleta (esquerda) e de *ranking* (direita).

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 [24].

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

2.1.4 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 [18]. 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.

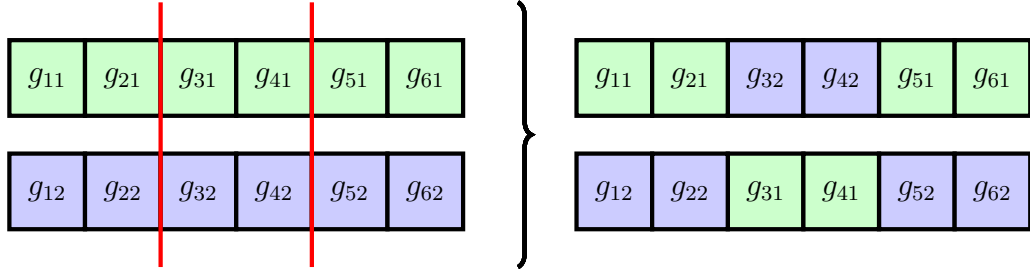


Figura 3: Recombinação em 2 pontos de corte gerando dois novos cromossomos.

Recombinação uniforme: Sejam $G_1 = (g_{11}, \dots, g_{n1})$ e $G_2 = (g_{12}, \dots, g_{n2})$ cromossomos obtidos na etapa de seleção. A recombinação uniforme passa por cada par de genes (g_{i1}, g_{i2}) , com $i = 1, \dots, n$, 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 – geralmente, $p = 0.5$, de modo que as escolhas sejam completamente aleatórias [22]. 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 [2].

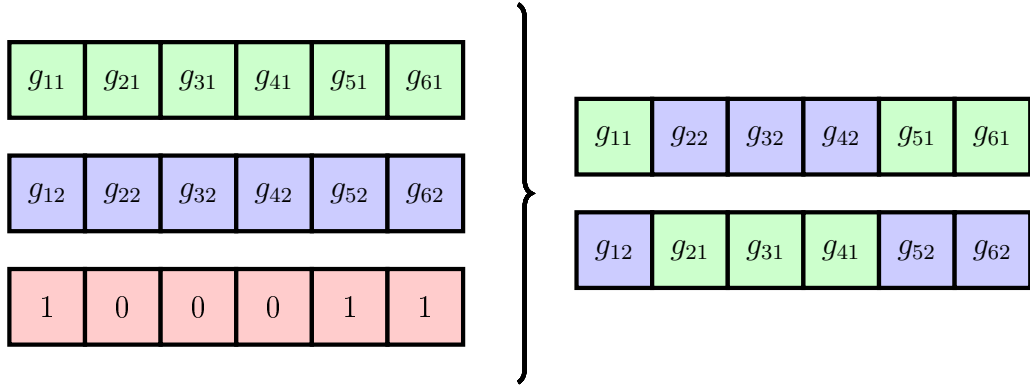


Figura 4: Recombinação uniforme de cromossomos usando máscara.

De acordo com Katoch, Chauhan e Kumar [14], 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 $G_1 = (g_{11}, \dots, g_{n1})$ e $G_2 = (g_{12}, \dots, g_{n2})$ 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 $k = i, \dots, j$, 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.

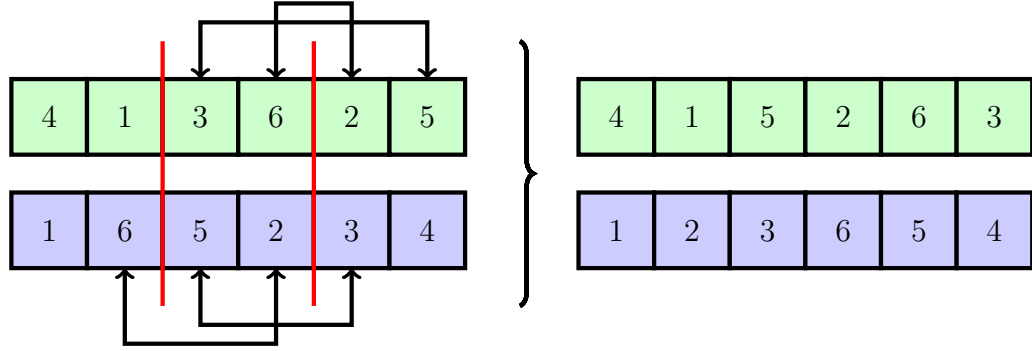


Figura 5: Recombinação de mapeamento parcial, ou PMX.

Recombinações baseadas em ordem: Para determinados problemas, como o problema do caixeiro viajante [21], é desejável manter a ordem de certos cromossomos intacta. Nestes casos, utilizam-se recombinações baseadas em ordem. Segundo Sastry, Goldberg e Kendall [22], o método de Davis [9] 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.

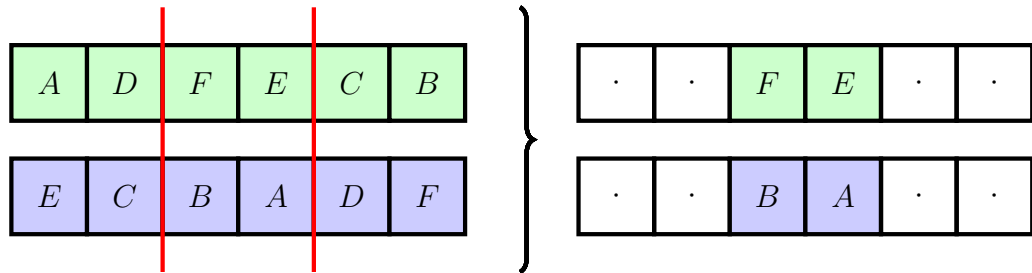


Figura 6: Primeiro passo de uma recombinação baseada em ordem.

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.

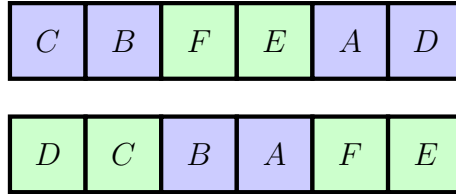


Figura 7: Segundo passo de uma recombinação baseada em ordem.

2.1.5 Mutação

O processo de mutação visa adicionar aleatoriedade aos cromossomos gerados por recombinação, possibilitando a exploração de soluções que não poderiam ser obtidas partindo apenas dos cromossomos selecionados para cruzamento. Soni e Kumar [25] apresentam nove tipos de mutação, alguns dos quais são exemplificados a seguir.

Mutação por inserção: Dois alelos aleatórios são selecionados. Um dos alelos é movido imediatamente à direita do outro, e os demais alelos têm suas posições trocadas conforme necessário. Esta técnica tem o potencial de mudar grande parte do cromossomo, por isso é mais interessante para resolver problemas de permutação, em que a ordem dos dados é mais importante do que a localização individual em si. A Figura 8 ilustra o procedimento.

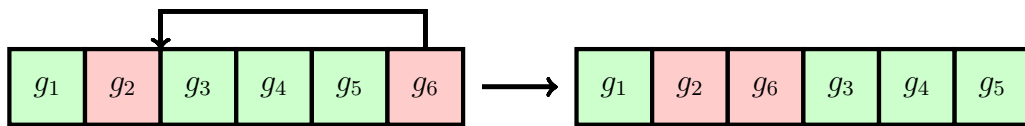


Figura 8: Mutação por inserção (g_2 e g_6 selecionados).

Mutação por inversão: Outra mutação interessante para problemas de permutação, a mutação por inversão envolve escolher dois alelos aleatórios do cromossomo e então inverter a ordem dos alelos entre eles, como mostra a Figura 9.

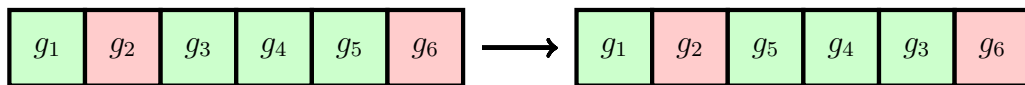


Figura 9: Mutação por inversão (g_2 e g_6 selecionados).

Mutações *flip*, *creep* e *uniforme*: Para muitos problemas, é suficiente representar dados utilizando genes binários. Neste caso, a mutação *flip* consiste em trocar genes valendo 0 por 1, e vice-versa. A mutação *creep* permite trocar genes do conjunto dos números reais por outros números reais, e a *uniforme* estende esta possibilidade para números inteiros. A Figura 10 ilustra o caso da mutação *flip*.

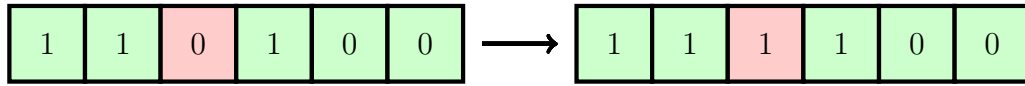


Figura 10: Mutação *flip* (g_3 selecionado).

2.1.6 Substituição

A etapa de substituição envolve atualizar a população com os cromossomos gerados nos processos de recombinação e mutação. Massago [16] destaca dois tipos principais de substituição: geracional e *steady-state*. Na substituição geracional, todos os indivíduos da população atual são substituídos pelos novos indivíduos, com a possível exceção de alguns indivíduos denominados elite. Preservar uma elite da população anterior pode ser útil para garantir que as melhores, ou mesmo piores, soluções serão mantidas. Na substituição *steady-state*, os indivíduos gerados por recombinação só entram na população caso exista solução pior na população atual que eles possam substituir. Sastry, Goldberg e Kendall [22] fornecem uma descrição mais geral do *steady-state*, segundo a qual é possível escolher outros parâmetros de substituição de indivíduos da população.

Após o término da etapa de substituição, o AG deve checar se alguma condição foi atingida (número máximo de iterações alcançado, valor objetivo mínimo atingido, etc.). Se sim, o AG deve parar e retornar a população atual, ou a melhor solução encontrada, ou um histórico de soluções. Do contrário, volta-se para a etapa de avaliação.

3 Um problema prático

Para testar as características de AGs discutidas anteriormente, optou-se por implementar o AG descrito por Sedighi et al. [23]. 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 [5]. 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](#).

3.1 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 11](#) 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 12](#) foi usado para gerar este mapa.

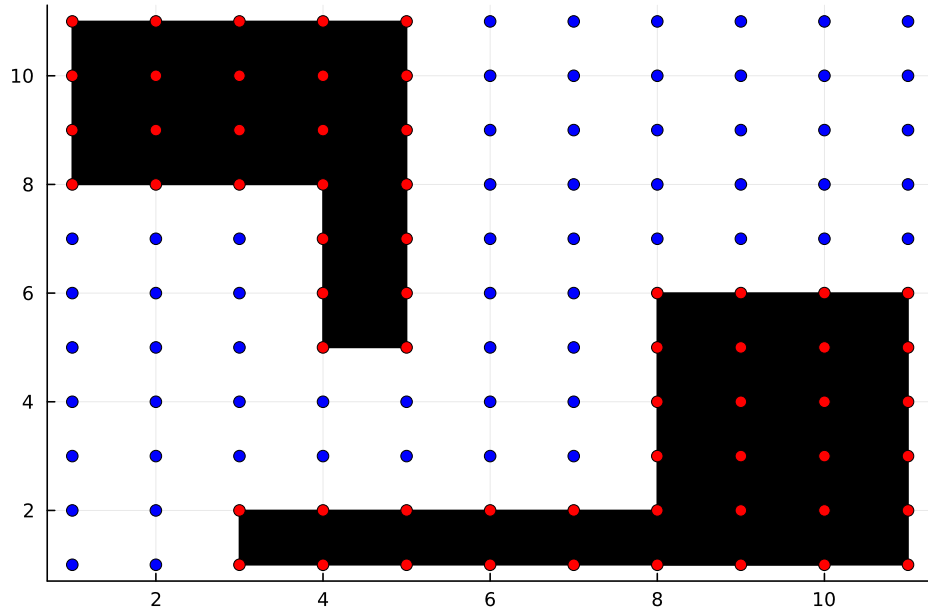


Figura 11: Exemplo de mapa discretizado.

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

Código 12: Código para geração de mapa.

3.2 Cromossomos e funções de aptidão

Os cromossomos propostos por Sedighi et al. [23] são divididos em quatro segmentos, como ilustra a Figura 13. 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,

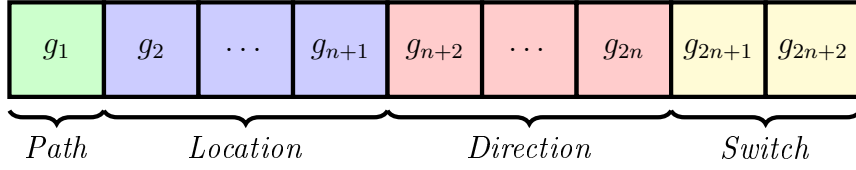


Figura 13: Cromossomo para roteirização [23].

se *location* for o vetor (x_1, \dots, x_n) , a sequência de deslocamentos *row-wise* do robô será $((1, x_1), (2, x_2), \dots, (n, x_n))$. 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 $((x_1, 1), (x_2, 2), \dots, (x_n, n))$.

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 14 e 15 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 $((1, 1), (2, 3), (3, 1), (4, 2), (5, 2), (6, 6))$ e $((1, 1), (3, 2), (1, 3), (2, 4), (2, 5), (6, 6))$.

As Figuras 14 e 15 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 [8]. 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.

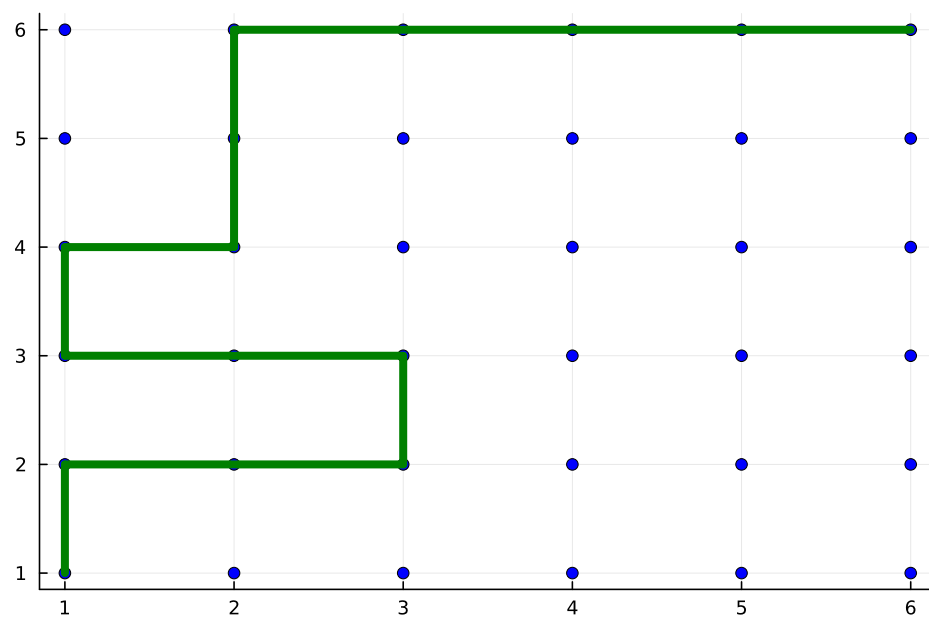


Figura 14: Deslocamento *row-wise*.

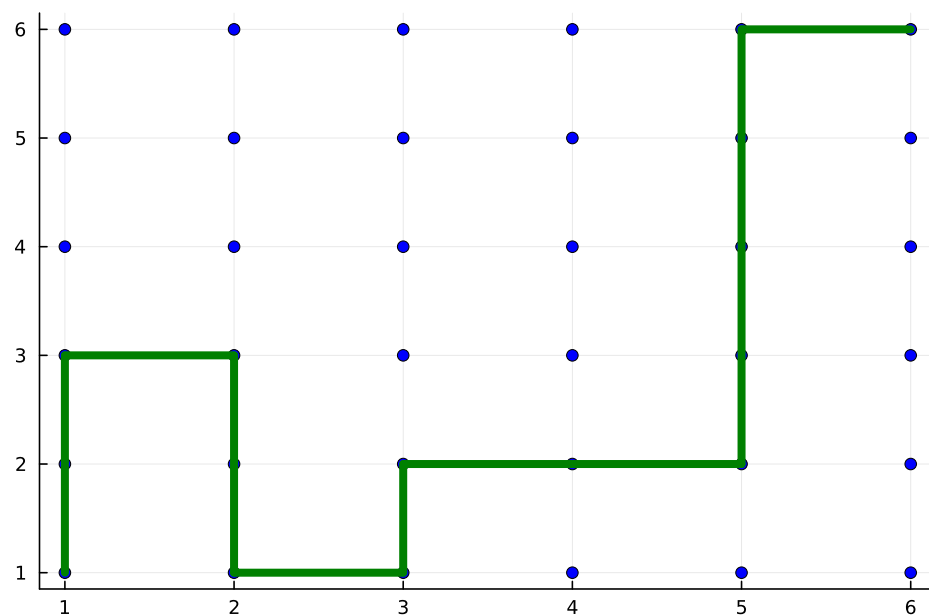


Figura 15: Deslocamento *column-wise*.

Referências

- [1] 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](https://doi.org/10.1016/j.ejco.2021.100011). URL: <https://www.sciencedirect.com/science/article/pii/S2192440621001386> (acesso em 05/06/2024).
- [2] 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](https://doi.org/10.11606/D.3.2000.tde-05092001-141334). URL: <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-05092001-141334/> (acesso em 10/06/2024).
- [3] Mokhtar S. Bazaraa, John J. Jarvis e Hanif D. Sherali. *Linear Programming and Network Flows*. 4^a ed. Wiley, 2010.
- [4] Dimitris Bertsimas e John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Dynamic Ideas, 1997.
- [5] Jeff Bezanson, Alan Edelman, Stefan Karpinski e Viral B. Shah. *Julia: A Fresh Approach to Numerical Computing*. 2015. DOI: [10.48550/arXiv.1411.1607](https://doi.org/10.48550/arXiv.1411.1607). arXiv: [1411.1607 \[cs\]](https://arxiv.org/abs/1411.1607). URL: <http://arxiv.org/abs/1411.1607> (acesso em 11/06/2024).
- [6] André Ponce de Leon F. de Carvalho. *Algoritmos Genéticos*. URL: <https://sites.icmc.usp.br/andre/research/genetic/> (acesso em 05/06/2024).
- [7] Pedro Henrique Centenaro. *Fundamentos Matemáticos e Computacionais de Problemas de Roteamento de Veículos*. 2023. URL: <https://repositorio.ufsc.br/handle/123456789/250623>.
- [8] Sulamita Maria Comini César e Dra Eliane Scheid Gazire. “MINICURSO DE GEOMETRIA TÁXI”. Em: (2010). URL: http://www1.pucminas.br/imagedb/documento/DOC_DSC_NOME_ARQUI20150306113049.pdf.
- [9] 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).

- [10] 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](https://doi.org/10.1007/978-3-319-91086-4_1). URL: http://link.springer.com/10.1007/978-3-319-91086-4_1 (acesso em 05/06/2024).
- [11] 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](https://doi.org/10.1007/978-1-4419-1665-5). URL: <https://link.springer.com/10.1007/978-1-4419-1665-5> (acesso em 03/06/2024).
- [12] 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](https://doi.org/10.1016/0305-0548(80)90001-5). URL: <https://linkinghub.elsevier.com/retrieve/pii/0305054880900015> (acesso em 06/04/2024).
- [13] Denny Hermawanto. “Genetic Algorithm for Solving Simple Mathematical Equality Problem”. Em: (2013).
- [14] 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](https://doi.org/10.1007/s11042-020-10139-6). URL: <http://link.springer.com/10.1007/s11042-020-10139-6> (acesso em 17/04/2024).
- [15] 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](https://arxiv.org/abs/2303.04147) [cs, math]. URL: <http://arxiv.org/abs/2303.04147> (acesso em 05/06/2024).
- [16] Sadao Massago. “Introdução ao Algoritmo Genético”. Em: (2013).
- [17] 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](https://doi.org/10.1038/srep34851). URL: <https://www.nature.com/articles/srep34851> (acesso em 05/06/2024).
- [18] 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](https://doi.org/10.17771/PUCRio.acad.28372). URL: http://www.maxwell.vrac.puc-rio.br/Busca_etds.php?strSecao=resultado&nrSeq=28372@1 (acesso em 10/06/2024).

- [19] Ademir Alves Ribeiro e Elizabeth Wegner Karas. *Otimização Contínua: Aspectos Teóricos e Computacionais*. 1ª ed. Curitiba: Editora Cengage, 2013.
- [20] Olympia Roeva, Stefka Fidanova e Marcin Paprzycki. “Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modeling”. Em: (2013).
- [21] Amanur Rahman Saiyed. “The Traveling Salesman Problem”. Em: (2012).
- [22] “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.
- [23] 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](https://doi.org/10.1109/CEC.2004.1331052).
- [24] 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](https://doi.org/10.1109/ABLAZE.2015.7154916). URL: <https://ieeexplore.ieee.org/abstract/document/7154916> (acesso em 06/06/2024).
- [25] Nitasha Soni e Tapas Kumar. “Study of Various Mutation Operators in Genetic Algorithms”. Em: 5 (2014).
- [26] James Stewart. *Cálculo*. 8ª ed. Vol. 1. Cengage Learning, 2016.
- [27] Saneh Lata Yadav e Asha Sohal. “Comparative Study of Different Selection Techniques in Genetic Algorithm”. Em: *International Journal of Engineering* 6.3 (2017).
- [28] Xiaolan Zhang. “Linear Programming”. Em: (2020). URL: https://storm.cis.fordham.edu/~zhang/cs5835/slides/LinearProgramming_handout.