Inteligência Artificial - Notas de aula

Raoni F. S. Teixeira

Aula 3 - Busca Local

1 Introdução

Nesta aula, estudaremos algoritmos que buscam bons estados sem examinar exaustivamente todo o ambiente. Em vez de explorar todas as possibilidades, esses algoritmos percorrem apenas partes do espaço de busca, movendo-se entre estados vizinhos até encontrar uma solução satisfatória — ou, idealmente, ótima.

Se cada estado ss estiver associado a uma pontuação $f(s)$, o objetivo do agente é encontrar o estado com o maior valor de f:

$$\underset{s \in \mathcal{S}}{\text{argmax}}\ f(s). \tag{1}$$

Esse processo é conhecido como **busca local**.

A busca local é amplamente usada em Inteligência Artificial e otimização, especialmente quando o espaço de estados é muito grande para ser explorado de forma completa. Os algoritmos apresentados aqui — Subida da Encosta, Têmpera Simulada e Algoritmos Genéticos — seguem essa abordagem, mas com estratégias distintas para escapar de ótimos locais e ampliar a exploração.

Um dos principais desafios desses métodos é equilibrar **exploração** (explore) e **refinamento** (exploit). Explorar significa investigar novas regiões do espaço, mesmo que isso leve a soluções temporariamente piores. Refinar significa concentrar-se nas melhores soluções já encontradas. A Subida da Encosta favorece o refinamento. Já a Têmpera Simulada e os Algoritmos Genéticos incorporam mecanismos explícitos de exploração, o que aumenta a chance de alcançar ótimos globais.

As próximas seções apresentam esses três métodos, com exemplos e pseudocódigos. Outros algoritmos relacionados podem ser encontrados em [RN09, KW19].

2 Subida da encosta

O algoritmo de subida da encosta é um dos métodos mais antigos de busca local, com raízes em otimização matemática. Seu uso na Inteligência Artificial remonta às primeiras abordagens de resolução de problemas com operadores, como nos sistemas GPS (General Problem Solver) desenvolvidos por Newell e Simon nos anos 1950 [NS59].

Pense em escalar uma montanha para alcançar o pico mais alto. A cada passo, você avalia os picos vizinhos e escolhe aquele com a maior elevação, até que não existam opções melhores.

Essa é a essência do algoritmo de subida da encosta. Ele começa em um estado aleatório s e, a cada passo, move-se para o vizinho com a maior pontuação, analisando apenas os estados próximos. O processo termina quando atinge um ótimo ou quando o número máximo de iterações é alcançado.

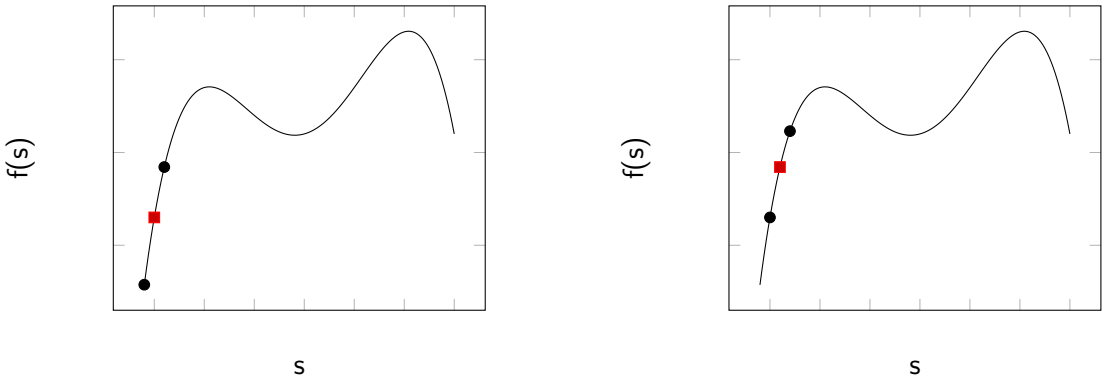
A Figura 1 mostra o funcionamento do algoritmo para uma função com dois máximos. A cor vermelha indica o estado solução (s) e a cor preta mostra os vizinhos imediatos. A cada iteração, s se move para o estado vizinho com a melhor pontuação. Quando o algoritmo termina (Figura 1f), a pontuação de s é maior que a pontuação de todos os seus vizinhos — máximo local.

O pseudo-código a seguir apresenta esse algoritmo usando uma função VIZINHANCA que devolve os vizinhos de um estado.

```
SUBIDAENCOSTA(f, max-it)
 1  s ← estado aleatório
 2  it ← 0
 3  while it < max-iter do
 4       melhor-vizinho ← s
 5       for each estado u ∈ VIZINHANCA(s) do
 6            if f(u) > f(melhor-vizinho) then
 7                 melhor-vizinho ← u
 8       if f(melhor-vizinho) > f(s) then
 9            s ← melhor-vizinho
10       else
11            break
12       it ← it + 1
13  return s
```

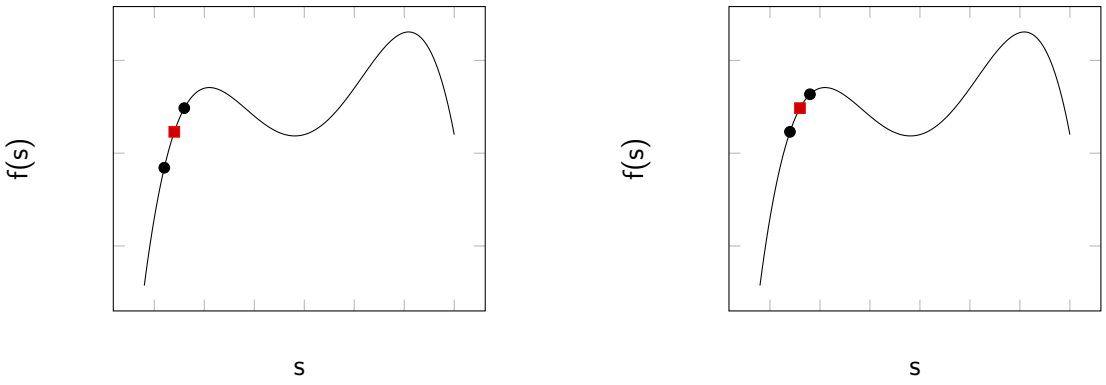
Para aplicar esse algoritmo a problemas reais, você precisa definir três elementos:

1. **Representação dos estados:** como os estados serão descritos?



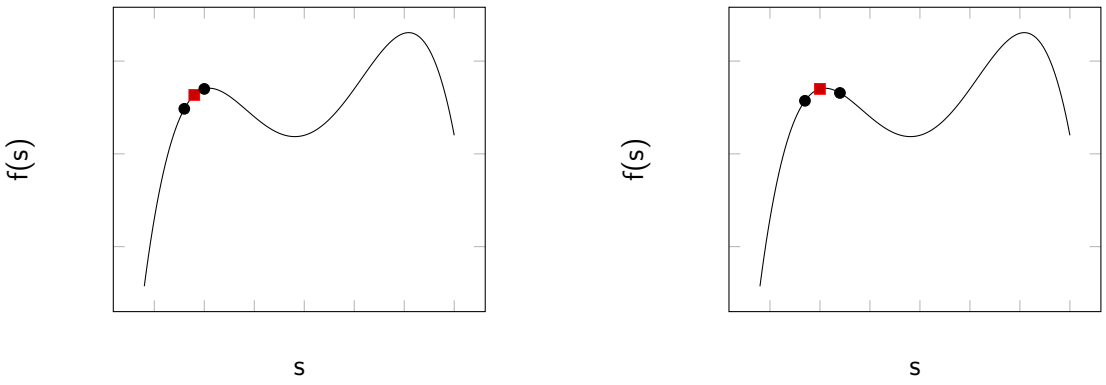
(a) Iteração 1.

(b) Iteração 2.



(c) Iteração 3.

(d) Iteração 4.



(e) Iteração 5.

(f) Iteração 6.

Figura 1: Seis iterações do algoritmo subida da encosta para um função f com dois mínimos. As cores vermelha e preta indicam respectivamente, estados em análise e vizinhos. A cada iteração, a solução se move para o estado vizinho com a melhor pontuação.

2. **Vizinhança:** quais estados próximos serão considerados?
3. **Função objetivo f:** como avaliar a qualidade de um estado?

A principal limitação do algoritmo de subida da encosta é sua tendência a ficar preso em ótimos locais. Para mitigar esse problema, é comum adotar a estratégia de reinício aleatório. Quando a busca atinge um ponto onde nenhum vizinho melhora a solução, o algoritmo é reiniciado a partir de um novo estado aleatório. O processo se repete por um número fixo de vezes ou até que uma solução satisfatória seja encontrada. Essa abordagem simples aumenta a cobertura do espaço de busca sem adicionar muita complexidade.

O Roteiro 3 mostra um exemplo de definição para o problemas da *N-Damas*.

3 Tempera Simulada

O algoritmo de Tempera Simulada (*Simulated Annealing*) é uma técnica inspirada no processo de recozimento de metais, onde o material é aquecido e, em seguida, resfriado de forma controlada. O método é utilizado para encontrar soluções aproximadas em problemas de otimização, especialmente quando o espaço de busca possui múltiplos ótimos locais.

A Têmpera Simulada foi proposta por [KGV83] como uma analogia ao recozimento físico de metais, sendo posteriormente adaptada para problemas de otimização combinatória. Seu uso em Inteligência Artificial começou a se popularizar nos anos 1980 em problemas como o caixeiro viajante e design de circuitos.

Ao contrário da Subida da Encosta, que pode ficar preso em ótimos locais, a Tempera Simulada permite movimentos que pioram a solução temporariamente, aumentando a chance de encontrar o ótimo global.

A cada iteração, o algoritmo escolhe aleatoriamente um vizinho do estado atual. Se o vizinho apresentar uma pontuação melhor, ele é aceito como o novo estado. Caso contrário, o algoritmo pode aceitá-lo com base em uma probabilidade que depende da temperatura atual e da diferença de pontuação entre os estados.

Essa probabilidade é dada pela fórmula:

$$P(\Delta E, T) = e^{-\Delta E/T},$$

em que ΔE é a diferença de pontuação ($f(\text{vizinho}) - f(\text{atual})$) e T é a temperatura atual.

A Figura 2 exibe o gráfico da função P para diferentes valores de ΔE, variando T. As curvas preta, azul e vermelha correspondem a ΔE igual a 20, 40 e 80, respectivamente. Em

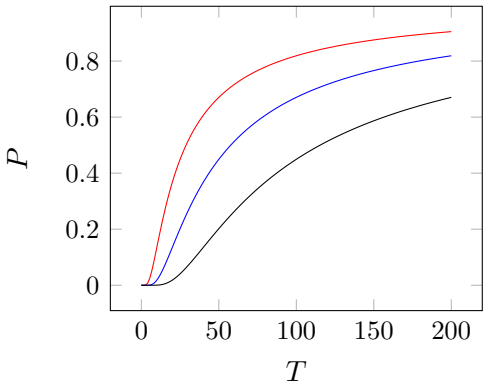


Figura 2: Probabilidade P em função da temperatura T para diferentes valores de ΔE. As curvas preta, azul e vermelha correspondem a $\Delta E = 20$, 40 e 80. À medida que T diminui, P também reduz, tornando o algoritmo mais conservador na aceitação de estados piores.

todos os casos, conforme a temperatura T diminui, a probabilidade de aceitar estados piores P também se reduz, tornando o algoritmo mais conservador.

A forma como a temperatura diminui ao longo das iterações é definida pela **função de resfriamento**. A forma mais comum é exponencial:

$$T_{k+1} = \alpha \cdot T_k,$$

com $\alpha \in (0, 1)$, geralmente entre 0,90 e 0,99. Quanto maior o valor de α, mais lenta a redução da temperatura, permitindo mais exploração. Uma queda rápida (valores menores de α) acelera a convergência, mas pode levar a ótimos locais. Outras estratégias incluem o *resfriamento logarítmico*, com $T_k = T_0 / \log(k + c)$, ou métodos adaptativos, que ajustam a temperatura com base no progresso da solução.

A escolha da função de resfriamento afeta diretamente o equilíbrio entre *exploração* e *refinamento* (*explore* vs *exploit*), impactando a eficácia do algoritmo.

O algoritmo pode ser descrito da seguinte forma:

TEMPERASIMULADA(f, T_{inicial}, taxa-resfriamento, max-it)

```
1  s ← estado aleatório
2  T ← T_inicial
3  it ← 0
4  while it < max-iter do
5      vizinho ← SELECIONARVIZINHO(s)
6      ΔE ← f(vizinho) − f(s)
7      if ΔE > 0 then
8          s ← vizinho
9      else
10         if RANDOM(0, 1) < e^{−ΔE/T} then
11             s ← vizinho
12     T ← T × α
13     it ← it + 1
14 return s
```

Os parâmetros são:

- **Estado inicial** (s_{inicial}): Ponto de partida do algoritmo.
- **Temperatura inicial** (T_{inicial}): Define a probabilidade inicial de aceitar estados piores.
- **Taxa de resfriamento** (α): Controla a redução da temperatura a cada iteração.
- **Número máximo de iterações** (max_{iter}): Limita a execução do algoritmo.

A função RANDOM gera um número real no intervalo $[0, 1]$, e a função SELECIONARVIZINHO escolhe aleatoriamente um estado vizinho para continuar a busca. O Roteiro 3 apresenta um exemplo aplicado ao problema das N-Damas.

Tanto a subida da encosta quanto a têmpera simulada limitam a busca a uma única região do espaço. No entanto, a têmpera simulada aprimora a subida da encosta ao permitir escapes controlados de ótimos locais, aumentando as chances de encontrar a solução global. O próximo algoritmo, o Algoritmo Genético, adota uma abordagem diferente: em vez de focar em uma única região, ele explora múltiplos pontos do espaço de busca simultaneamente.

4 Algoritmo Genético

Algoritmo genético é um método iterativo de otimização que opera localmente sobre múlltiplas regiões do espaço de busca, combinando as informações dos estados dessas regiões para evitar minimos locais.

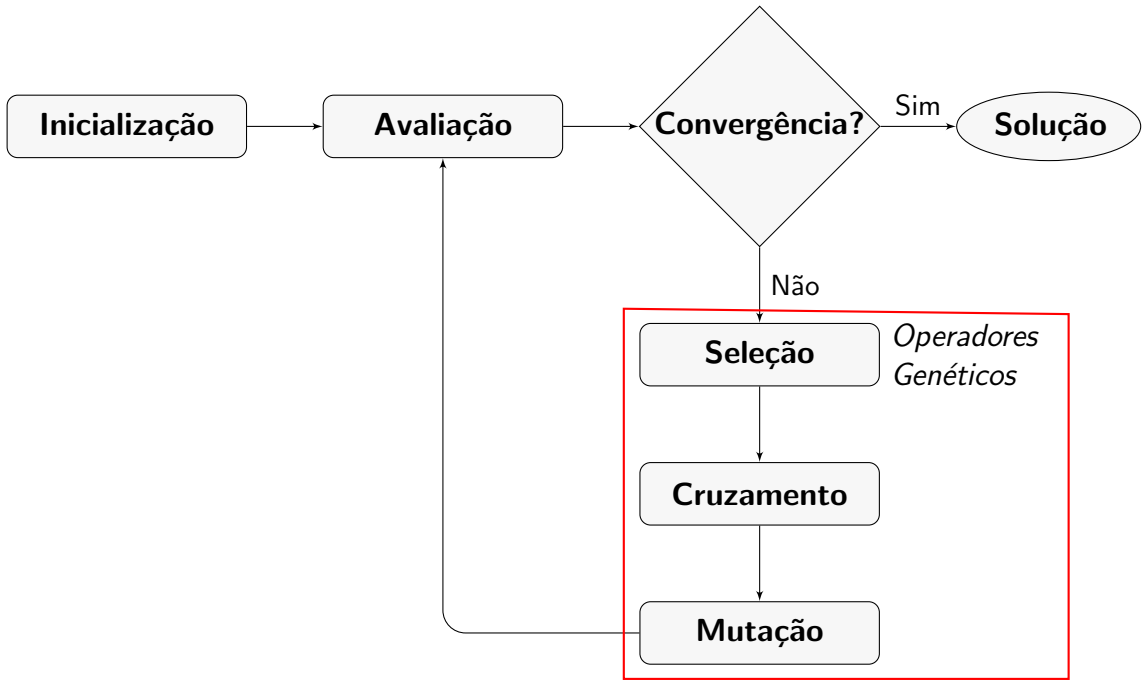


Figura 3: Algoritmo genético. O algoritmo termina quando um critério de convergência é atingido.

Algoritmos Genéticos foram introduzidos por John Holland na década de 1970 [Hol75] como uma abordagem computacional inspirada na evolução natural. Sua aplicação em Inteligência Artificial foi impulsionada por trabalhos como os de [Gol89], com uso em aprendizagem de máquinas, otimização e controle adaptativo.

Inspirado na biologia evolutiva, o conjunto de estados analisados é chamado de **população** e cada ciclo de iteração é uma **geração**. Cada estado é chamado de *indivíduo* e é representado por uma sequência de *cromossomos* — um vetor de números. Operações denominadas *seleção*, *cruzamento* e *mutação* produzem uma nova população combinando e alterando os vetores da população atual.

O algoritmo consiste nos passos do fluxograma da Figura 3:

- **Inicialização:** define uma população inicial composta por M indivíduos gerados aleatoriamente. Essa população deve estar bem distribuída pelo espaço de busca, aumentando as chances de encontrar uma solução. Dois parâmetros são definidos nesta etapa: (a) a distribuição utilizada na amostragem e (b) o número de amostras M. Um valor maior de M melhora a representatividade do espaço, mas também aumenta o consumo de memória.
- **Avaliação:** cada indivíduo da população é avaliado por uma função f, que retorna um

número real representando sua qualidade. Seguindo a inspiração biológica, a função de avaliação é chamada de *fitness*. O algoritmo termina quando encontra um indivíduo ótimo ou atinge o número máximo de iterações.

- **Seleção:** escolhe os indivíduos mais bem avaliados para atuarem como pais da próxima geração. A seleção é realizada aleatoriamente, mas com maior probabilidade para indivíduos com melhores valores de f.
- **Cruzamento:** combina os cromossomos dos indivíduos selecionados para criar novos descendentes. A Figura 4 ilustra essa operação em torno de um ponto de cruzamento (k), um número sorteado aleatoriamente no intervalo $[0, N[$. O cruzamento concatena os vetores $v[0 \dots k]$ e $v[k+1 \dots N-1]$ dos pais, formando os descendentes.
- **Mutação:** altera aleatoriamente o valor de algum cromossomo de um indivíduo. A taxa de mutação define a probabilidade de essa alteração ocorrer. Uma taxa maior aumenta a chance de mutação. Tanto a posição do cromossomo quanto seu novo valor são determinados de forma aleatória e uniforme.

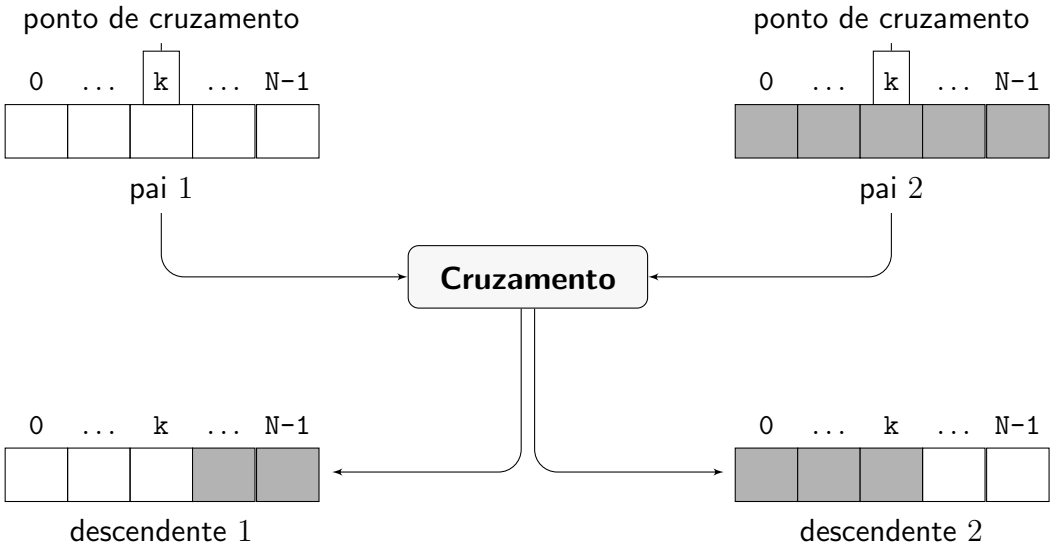


Figura 4: Operação de cruzamento. Os descendentes são uma combinação dos pais.

A aplicação dos operadores genéticos exige duas definições fundamentais: uma representação vetorial para os estados (como vetores de números ou bits) e uma função de avaliação que mede a qualidade de cada solução. O exemplo disponível no Roteiro 3 mostra como aplicar essas ideias na prática.

Em algoritmos genéticos, cada solução é representada por um vetor — o cromossomo. A escolha dessa representação depende do tipo de problema e pode assumir diferentes formas:

- **Binária**: usa sequências de 0s e 1s. É comum em problemas combinatórios.
- **Reais**: emprega números reais, úteis em otimizações contínuas.
- **Inteiros**: adequada para problemas de alocação ou roteamento.
- **Estruturas complexas**: em variantes como a programação genética, a solução pode ser representada por árvores ou grafos.

A representação escolhida deve ser compatível com os operadores de cruzamento e mutação, pois afeta diretamente o desempenho do algoritmo.

Outro fator decisivo para o sucesso do algoritmo é a diversidade da população. Quando a população perde variedade, o algoritmo pode convergir para soluções ruins — a chamada convergência prematura. Para evitar isso, uma técnica comum é o elitismo: os melhores indivíduos da geração atual são mantidos na próxima geração, garantindo que boas soluções não se percam. A mutação e o cruzamento introduzem novas variações, mantendo a busca ativa em diferentes regiões do espaço.

De forma geral, um algoritmo genético é um processo iterativo que explora múltiplas regiões do espaço de busca em paralelo. A cada geração, os operadores genéticos produzem novas soluções a partir das anteriores. A semelhança entre gerações depende da intensidade da mutação: quanto maior o grau de aleatoriedade, maior a chance de explorar caminhos novos.

5 Discussão Final

Cada algoritmo apresenta pontos fortes e limitações. A escolha do método depende das características do problema, da precisão exigida e do tempo disponível (Veja Tabela 1). A Subida da Encosta funciona bem em problemas simples e bem definidos. A Têmpera Simulada é útil quando há muitos ótimos locais e é preciso escapar deles. O Algoritmo Genético, por sua vez, é indicado quando se deseja explorar simultaneamente várias regiões do espaço de busca.

6 Exercícios

1. Utilize o código do Roteiro 3 e compare o desempenho da Subida da Encosta, da Têmpera Simulada e do Algoritmo Genético no problema das N-damas (por exemplo, N

Tabela 1: Comparação entre Subida da Encosta, Têmpera Simulada e Algoritmo Genético

Característica	**Subida da Encosta**	**Têmpera Simulada**	**Algoritmo Genético**
Determinismo	Determinístico	Estocástico	Estocástico
Região de busca	Local	Local (com escapes)	Múltiplas regiões
Aceita soluções piores	Não	Sim (com probabilidade)	Sim (via diversidade)
Representação	Vetorial	Vetorial	Vetorial (flexível)
Requer população	Não	Não	Sim
Vulnerável a ótimos locais	Sim	Menos	Menos
Custo computacional	Baixo	Moderado	Alto

= 30). Meça o tempo de execução, a taxa de sucesso e o número médio de iterações. Com base nos resultados, discuta qual método é mais apropriado e por quê.

Referências

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[KGV83] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KW19] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. The MIT Press, 2019.

[NS59] Allen Newell and Herbert A. Simon. The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1959.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.