Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Otimização e Algoritmos Genéticos

Notas de Aula

Vítor Wilher

OAG · Eletiva

Versão 1.0

Resumo. Métodos de otimização e algoritmos evolutivos aplicados a problemas de busca e decisão.
Palavras-chave: otimização; algoritmos genéticos; metaheurísticas

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Otimização por Algoritmos Genéticos (OAG)** trata de uma das técnicas centrais da Inteligência Computacional: a otimização evolucionária. O ponto de partida é uma constatação prática — otimizar é essencialmente melhorar a solução de problemas em negócios, na indústria e em processos operacionais, o que pode trazer eficiência e rentabilidade, redução de despesas, gastos e perdas, e aumento dos lucros. O problema é que muitos desses problemas de otimização são complexos, têm grandes espaços de busca e são de difícil modelagem. É nesse território que os Algoritmos Genéticos (AGs) se mostram úteis.

Conforme o material da primeira aula, o curso visa, além de introduzir os conceitos fundamentais de otimização, apresentar teoria e prática da técnica de otimização evolucionária conhecida como Algoritmos Genéticos. A sequência de tópicos é cumulativa e segue a própria anatomia de um AG: primeiro o ciclo evolutivo em alto nível; depois a representação, a decodificação e a avaliação de soluções; em seguida a reprodução genética (seleção, cruzamento e mutação) com as técnicas de controle da evolução; depois o tratamento de restrições; e, por fim, a avaliação e otimização com múltiplos objetivos. Essa progressão não é arbitrária. Os slides insistem em um ponto que atravessa todo o curso: os AGs são flexíveis e permitem a fácil inclusão de instruções específicas para o problema de interesse, mas **a qualidade dos resultados depende diretamente da qualidade da modelagem do problema** — isto é, da representação cromossômica e decodificação, da função de avaliação e dos operadores genéticos.

Na prática, o material apoia-se em exercícios recorrentes — o problema das quatro rainhas, a maximização de funções polinomiais simples, a função F6, a localização de antenas de telecomunicações, o mix de produção de uma fábrica e a alocação de área de plantio — resolvidos ora em planilha (Excel, Solver, Evolver), ora em Python com a biblioteca DEAP. A avaliação combina participação com quizzes a cada aula, além de talks e prova. A bibliografia sugerida é: *Algoritmos Genéticos: Uma importante ferramenta da Inteligência Computacional* (Ricardo Linden, 2006), *Genetic Algorithms + Data Structures = Evolution Programs* (Michalewicz, Z., 1996) e *Genetic Algorithms in search, optimization and machine Learning* (David E. Goldberg, 1989). Os professores indicados no material são Ana Carolina Abreu e Felipe Borges.

2 Introdução aos Algoritmos Genéticos

2.1 O que é otimizar e quais são suas entidades

O objetivo da otimização, na definição dada em aula, é **buscar a solução ótima ou melhorar a solução que se possui**. Ela é essencial para aumentar o desempenho e a eficácia de qualquer processo e para aumentar a competitividade nos negócios. Um requisito levantado nos slides é que a otimização deve ser prática e flexível, de modo a ser empregada em qualquer processo, a qualquer tempo — ou seja, dinâmica.

Todo problema de otimização pode ser descrito por seis entidades: (1) o **problema**, com suas características, restrições e variáveis; (2) as **variáveis**, cujos valores afetam a qualidade da solução; (3) a **função objetivo**, que mede o quão boa é uma solução; (4) o **método, algoritmo ou heurística** de busca; (5) o **espaço de busca**, isto é, o número total de soluções, determinado pelo número de variáveis e seus domínios; e (6) os **recursos computacionais** para processamento do método, avaliação das soluções e escolha da melhor solução.

O exemplo didático usado para fixar essas entidades é a **cabra cega**: a busca de um tesouro escondido em uma área. O problema é a brincadeira; o método é aleatório somado a um componente intelectual do tipo "se isso então aquilo"; as variáveis são `x` e `y`, a posição em uma área; a função objetivo é o retorno "tá frio", "tá morno", "tá quente", que corresponde à distância euclidiana ao tesouro; o espaço de busca é a área, com número de soluções dado pelo produto dos domínios de `x` e `y`; e o recurso computacional é o cérebro do jogador.

Esse exemplo introduz uma ideia essencial: **a avaliação adapta a busca**. Ao testar `(x0, y0)` e receber "tá frio", depois `(x1, y1)` e receber "tá morno", e então `(x2, y2)` e receber "tá quente", a informação corrente reorienta as tentativas seguintes. Quando a área é muito grande, a solução é a **busca paralela**: vários pontos testados ao mesmo tempo, com a possibilidade de combinar coordenadas de dois pontos promissores — gerar `(xB, yA)` e `(xA, yB)` a partir de A e B. Essa combinação é exatamente o **cruzamento**.

2.2 Inteligência Artificial e conceitos básicos dos AGs

Os métodos mais comuns de otimização citados são a busca aleatória (exaustiva, quando há tempo), a busca aleatória combinada com heurísticas e bom senso, e os métodos e algoritmos científicos. Entre estes entram as técnicas de Inteligência Artificial, também chamada de Inteligência Computacional e definida como o conjunto de "técnicas, modelos e sistemas computacionais que imitam aspectos humanos e naturais que incorporam inteligência: percepção, raciocínio, aprendizado, evolução e adaptação". As técnicas listadas e suas inspirações são: **Sistemas Especialistas** (inferência humana), **Lógica Fuzzy** (processamento linguístico), **Redes Neurais** (neurônios biológicos), **Algoritmos Genéticos** (evolução biológica) e **Sistemas Híbridos** (aspectos combinados). Suas aplicações incluem suporte à decisão, reconhecimento de padrões, previsão, otimização, controle, modelagem, planejamento, detecção de fraude e descoberta de conhecimento (data mining).

Os Algoritmos Genéticos são algoritmos baseados nos mecanismos de seleção natural e genética, inspirados no Princípio da Evolução das Espécies proposto por Darwin: "quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes." A analogia com a natureza é sistematizada assim: indivíduo corresponde a solução; cromossoma, a representação; reprodução sexual, ao operador cruzamento; mutação, ao operador mutação; população, a conjunto de soluções; gerações, a ciclos; e meio ambiente, ao problema.

Os AGs empregam um processo **adaptativo e paralelo** de busca em problemas complexos. Adaptativo porque a informação corrente influencia a busca futura; paralelo porque várias soluções são consideradas a cada momento; e o problema é dito complexo quando é de difícil formulação matemática ou possui grande espaço de busca. A ilustração da complexidade usa o problema de maximizar `f(x) = x^2` com `x` entre 0 e $2^L - 1$: à medida que o número de bits `L` cresce, o número de soluções cresce exponencialmente e mesmo uma máquina de 10^9 instruções por segundo se torna insuficiente para a busca exaustiva.

Em resumo, nos AGs cada indivíduo representa uma possível solução; uma população é criada e submetida a seleção, cruzamento e mutação; a qualidade de cada indivíduo é determinada por sua avaliação; e gera-se um processo de evolução natural que eventualmente deverá produzir um indivíduo que caracterize uma boa solução — talvez até a melhor — para o problema.

2.3 Representação, operações e o ciclo de otimização

A **representação** consiste em traduzir a informação do problema para uma forma viável de ser tratada pelo computador; quanto mais adequada ao problema, maior a qualidade dos resultados. A estrutura básica é o **cromossomo**, composto por **genes**.

As operações fundamentais são: **seleção**, que privilegia os indivíduos mais aptos; **reprodução**, em que os indivíduos são reproduzidos com base na aptidão; **cruzamento**, a troca de genes (pedaços de palavras); e **mutação**, a troca aleatória de um gene (bit da palavra).

O exemplo canônico é achar o valor máximo de `f(x) = x^2` com `x` entre 0 e 63, usando palavras binárias que representam sucessivas potências de 2: `011100` representa 28 e `110101` representa 53, uma solução mais apta. Com quatro cromossomos:

Cromossoma	Palavra	x	Aptidão
A	100100	36	1296
B	010010	18	324
C	010110	22	484
D	000001	1	1

A regra de seleção é que a **probabilidade de seleção é aproximadamente proporcional à aptidão do cromossoma**.

Os operadores genéticos têm a função de modificar os indivíduos e gerar novos. O **cruzamento** recombina o material genético de dois indivíduos a fim de criar dois novos, extraindo genes de indivíduos diferentes; a forma introdutória é a de **1 ponto de corte**, mas existem outras, como 2 pontos de corte, uniforme e baseado em maioria. A **mutação** introduz diversidade na população, sendo responsável pela variação dos indivíduos, e consiste em aplicar modificações aleatórias em uma ou mais características de um indivíduo. Utilizando ambos, os AGs conseguem um equilíbrio entre a capacidade de **exploração** do espaço de soluções e o **aproveitamento** das melhores soluções ao longo da evolução.

O ciclo de otimização é: a partir da população atual, selecionam-se **pais**; aplica-se a **reprodução** (crossover e mutação); obtêm-se os **filhos**; realiza-se a **avaliação dos filhos** por `f()`; e o processo se repete, caracterizando a **evolução**. Como vantagens, o material aponta que os AGs são uma técnica de busca global (evita mínimos locais), servem à otimização de problemas complexos e mal estruturados e dispensam formulação matemática precisa. Como pontos de atenção: exigem precisão na representação do cromossoma, a evolução pode ser demorada em alguns problemas e a modelagem depende da habilidade do especialista.

As áreas de aplicação citadas são Energia, Finanças, Engenharia e Telecomunicações, Medicina, Meio Ambiente, Indústria e Comércio. A atividade de Breakout Room pede a pesquisa de três exemplos de aplicação. Exemplos típicos mencionados incluem otimização de produção; roteiro de entrega de cargas visando redução de custos e antecipação de entregas; campanhas de marketing; planejamento e programação (scheduling) industrial; cadeia de suprimentos; design e dimensionamento de estruturas; otimização de portfólio; fluxo de caixa; e alocação de recursos humanos em projetos.

3 Representação, Decodificação, Avaliação de Soluções

3.1 Componentes e representação

A aula organiza o AG em componentes que servirão de roteiro para todo o curso: 1. Problema; 2. Representação; 3. Decodificação; 4. Avaliação; 5. Operadores; 6. Técnicas; 7. Parâmetros. O **estudo de contexto do problema** vem primeiro: conhecer regras, restrições, objetivos e procedimentos em uso. Os AGs são indicados em problemas difíceis de otimização, com muitos parâmetros e variáveis, mal estruturados (condições e restrições difíceis de modelar matematicamente) e com grandes espaços de busca em que não é possível a busca exaustiva.

A representação é fundamental e deve: descrever o espaço de busca relevante ao problema; codificar geneticamente a "essência" do problema; e ser compatível com os operadores de crossover e mutação. A escolha depende do tipo de problema: problemas **numéricos** admitem representação binária, real ou inteira; problemas de **ordem**, lista; problemas de **grupo**, vetor; problemas **mistos**, representação mista.

A **representação binária** foi o primeiro tipo usado em AGs. Um número real é codificado por um número binário de `K` bits, descrevendo o real em detalhes (os genes). O exemplo elementar é 13 em binário, `1101`, que equivale a $1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$. Suas qualidades: representa números na menor base (2), é simples de criar e manipular, produz bons resultados e tem decodificação numérica fácil. A ressalva é que facilita a demonstração, porém nem sempre é adequada.

3.2 Decodificação

Decodificar é construir a solução do problema a partir de um cromossoma. Cromossomas "representam" soluções, e a decodificação é a etapa que permite que cada indivíduo seja efetivamente avaliado. Para `0011011`, tem-se $0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 27$.

O exemplo trabalhado encadeia representação, decodificação e avaliação para `f = x^2 - 2x` sobre dez indivíduos de 4 bits: A (`1011`) decodifica para 11 e avalia 99; B (`1111`) para 15 e 195; C (`0010`) para 2 e 0; D (`1101`) para 13 e 143; E (`1000`) para 8 e 48; F (`0011`) para 3 e 3; G (`1110`) para 14 e 168; H (`1100`) para 12 e 120; I (`1010`) para 10 e 80; J (`0111`) para 7 e 35. Os exercícios repetem o roteiro com `f(x) = x^3 + 15x` e `f(x) = x^3 - 11x^2 + 3.2x + 1.9` sobre populações de 6 bits.

```python
def decodifica_binario(cromossomo):
    valor = 0
    for bit in cromossomo:
        valor = valor * 2 + bit
    return valor

def avalia(cromossomo):
    x = decodifica_binario(cromossomo)
    return x**2 - 2*x
```

O exemplo central de modelagem é o **problema das quatro rainhas**: dispor as rainhas em um tabuleiro 4 x 4 de forma que nenhuma seja atacada por outra, isto é, que duas rainhas quaisquer não estejam na mesma linha, coluna ou diagonal. A pergunta da aula não é qual a resposta ótima, mas **como modelar**: como seria o cromossomo, quantos genes teria e o que cada gene representaria. São apresentadas três abordagens: na **abordagem 1**, um bit para cada casa do tabuleiro, com

cromossomo de 16 genes onde 1 indica presença de rainha (por exemplo, `1010100010100001`); na **abordagem 2**, cada rainha descrita por um par linha-coluna codificado em binário, também com 16 bits (`0000010111100010`); e na **abordagem 3**, uma codificação mais compacta de 8 bits (`00000101`). A comparação é o coração pedagógico do exercício: representações diferentes descrevem o mesmo problema com espaços de busca de tamanhos muito distintos.

3.3 Avaliação, normalização e windowing

A **avaliação** é a maneira utilizada para determinar a qualidade de um indivíduo como solução do problema. A função de avaliação permite diferenciar entre boas e más soluções e deve embutir todo o conhecimento que se possui sobre o problema, assim como seus objetivos de qualidade. Um cuidado importante: problemas cuja solução é do tipo "tudo ou nada" devem ter sua avaliação modificada para introduzir certo **gradualismo**. No caso das quatro rainhas, pode-se verificar quantas rainhas satisfazem todas as restrições. Sem gradualismo, o AG não tem sinal para evoluir.

O **método clássico** consiste em atribuir como aptidão o valor numérico do resultado da avaliação. Embora muito utilizado, apresenta duas situações que precisam ser tratadas.

Superindivíduos são indivíduos com avaliação muito superior à média, capazes de dominar o processo de seleção; sua presença impede que o AG obtenha novas soluções potencialmente melhores. O exemplo usa `f(x) = x^2` com avaliações 256, 16, 9 e 1 (soma 286), gerando fatias de roleta de 90,8%, 5,7%, 3,2% e 0,3% — praticamente só um indivíduo se reproduz.

Competição próxima ocorre quando as aptidões são numericamente muito próximas, dificultando distinguir a qualidade dos indivíduos. O exemplo usa 999,979, 999,514 e 999,066, que produzem fatias de aproximadamente 33,35%, 33,33% e 33,32% — a seleção torna-se praticamente aleatória e a pressão evolutiva desaparece.

Para resolver ambos, existem técnicas de alteração da função de avaliação. A **normalização** consiste em dar valores às avaliações dentro de um intervalo determinado. No exemplo com 256, 16, 9 e 1, a normalização por ranking transforma fatias de 90,8%, 5,7%, 3,2% e 0,4% em 45,5%, 31,8%, 18,2% e 4,5%. As cinco equações apresentadas são:

1. Por posição no ranking: `V' = novo_min + (novo_max - novo_min) / (n - 1) * (i - 1)`
2. `V' = log2(v)`
3. `V' = log10(v)`
4. Min-max simples: `V' = (v - min) / (max - min)`
5. Min-max com reescala: `V' = (v - min) / (max - min) * (novo_max - novo_min) + novo_min`

Para os valores originais 256, 16, 9 e 1, a equação (1) produz 10, 7, 4 e 1; a (2), 8,0, 4,0, 3,2 e 0,0; a (3), 2,4, 1,2, 1,0 e 0,0; a (4), 1,00, 0,06, 0,03 e 0,00; e a (5), 10,00, 1,53, 1,28 e 1,00.

O **windowing** consiste em achar o valor mínimo dentre as avaliações da população e designar a cada cromossomo uma avaliação igual à quantidade que **excede** esse mínimo: `V' = v - v_min`. Há a variante com **aptidão mínima de sobrevivência**, `V' = v - v_min + AP_min`, que garante alguma chance ao pior indivíduo. No caso de competição próxima, os indivíduos com avaliações 999,979, 999,066 e 999,514 passam, com windowing, a 0,913, 0 e 0,448; e, com avaliação mínima, a 0,963, 0,05 e 0,498 — as roletas mudam de cerca de 34%, 33% e 33% para aproximadamente 64%, 3% e 33%. A pressão seletiva é restaurada.

3.4 Representação por números reais

Ao codificar reais em binário, os aspectos importantes são: as variáveis `(x1, ..., xt)`; o domínio, com `xi` em `(min_i, max_i)`; e a **precisão**, em `p` casas decimais. O número de soluções distintas é `(max_i - min_i) * 10^p`, e a precisão obtida com `ki` bits é `(max_i - min_i) / (2^ki - 1)`. A fórmula de decodificação é `x_real = x_bin * (max_i - min_i) / (2^ki - 1) + min_i`, de modo que uma cadeia só de zeros dá `min_i` e uma só de uns dá `max_i`.

O caso de estudo é a **função F6**, cujo objetivo é maximizar; ela possui uma única solução ótima, `F6(0,0) = 1`, e é difícil de otimizar por conter vários mínimos locais. A representação é binária codificando real, com duas variáveis no domínio `[-100, +100]` e precisão de 4 a 5 casas decimais, levando a `Ki = 22` bits por variável — **44 bits** no total. O exemplo completo: o cromossoma é dividido em duas metades de 22 bits; convertidas para base 10 dão 165377 e 2270139; multiplicadas por `200 / (2^22 - 1)` dão 7,885791751335085 e 108,24868875710696; somadas ao mínimo, dão `x = -92,11420824866492` e `y = 8,248688757106959`; aplicadas a F6, resultam em 0,5050708.

```
def decodifica_real(bits, minimo, maximo):
    k = len(bits)
    inteiro = int("".join(str(b) for b in bits), 2)
    return inteiro * (maximo - minimo) / (2**k - 1) + minimo
```

Para introduzir a representação real, a aula apresenta a **hibridização**. Um AG híbrido é construído a partir do "algoritmo de otimização em uso" no problema: `Algoritmo Híbrido = Algoritmo em Uso + Algoritmo Genético`. Hibridizar significa adotar a **representação** em uso, adaptar os **operadores** e adotar as **heurísticas** de otimização. As vantagens: incorpora o conhecimento no domínio do problema; resulta num sistema mais familiar para o usuário; e o algoritmo em uso pode fornecer "sementes" para o AG. No caso da F6, o algoritmo em uso é uma busca aleatória de `x` e `y` em planilha Excel; hibridizando, a representação passa a ser uma lista de reais `(x, y)`, a avaliação é `f6(x,y)`, a inicialização usa reais aleatórios e os operadores são crossover, mutação e operadores inspirados no problema.

Os operadores para representação real são:

- **Crossover de 1 ou 2 pontos, ou uniforme**, sobre a lista de reais. Com o padrão `0 1 1 0` e pais `P1 = (x1, y1, t1, z1)` e `P2 = (x2, y2, t2, z2)`, geram-se `F1 = (x2, y1, t1, z2)` e `F2 = (x1, y2, t2, z1)`. O material observa que esse crossover é pouco eficiente para problemas contínuos.
- **Crossover de média**: se dois cromossomas são promissores, a média de seus valores reais pode levar a uma melhor solução; de `P1` e `P2` gera-se `F1 = ((x1+x2)/2, (y1+y2)/2)`.
- **Crossover aritmético**: combinação linear dos genitores, `F1 = a*P1 + (1-a)*P2` e `F2 = a*P2 + (1-a)*P1`, com `a` aleatório em `[0, 1]`.
- **Mutação de real**: substitui cada número real por um real aleatório quando o teste de probabilidade resulta verdadeiro; tem **alto poder de dispersão**.
- **Mutação CREEP**: implementa uma **busca local**, ajustando aleatoriamente em ambas as direções. O método de ajuste 1 é `X(t+1) = X(t) + delta(max - X(t))` se o bit sorteado for 0, e `X(t+1) = X(t) - delta(X(t) - min)` se for 1, com `delta(s) = s * rand` e `rand` aleatório em `[0, p]`. Se `p` é pequeno o ajuste é menor; se grande, maior.

```python
import random

def crossover_media(p1, p2):
    return [(a + b) / 2 for a, b in zip(p1, p2)]

def crossover_aritmetico(p1, p2):
    a = random.random()
    return ([a*x + (1-a)*y for x, y in zip(p1, p2)],
            [a*y + (1-a)*x for x, y in zip(p1, p2)])
```

A comparação final entre binário e real: a representação por reais é mais adequada em domínios contínuos, sobretudo em grandes domínios em que a binária exigiria cromossomos muito longos (100 variáveis em `[-500, 500]` com 4 casas decimais exigiriam cerca de 2400 bits); é mais rápida, pois não há decodificação; oferece maior precisão; permite operadores específicos ao problema; e evita os **Hamming Cliffs**, pois dois pontos próximos no espaço de representação também estão próximos no espaço do problema. O exemplo da **distância de Hamming** ilustra o ponto: `C1 = 011111` (31) e `C2 = 100000` (32) têm distância 6 apesar de serem vizinhos em valor; na representação real, a distância entre 31 e 32 é 1.

4 Reprodução Genética: Seleção, Cruzamento e Mutação

4.1 O fluxo do processo evolutivo

Esta aula formaliza o AG como algoritmo completo:

```
início
  ler número de gerações
  ler tamanho da população
  ler taxas de operadores
  gerar P(t), com t = 0
  avaliar P(t)
  enquanto não atingir o máximo de gerações:
      selecionar P(t) a partir de P(t-1)
      cruzamento P(t)
      mutação P(t)
      avaliar P(t)
      t = t + 1
  retornar melhor indivíduo de P(t)
fim
```

Cada elemento é explicado: o **tamanho da população** define o número de soluções a serem avaliadas a cada geração; o **número de gerações** define a quantidade de ciclos de evolução, isto é, a condição de parada; as **taxas de operadores** são as probabilidades de aplicação dos operadores de cruzamento e mutação; **gerar P(t)** cria a população inicial; **avaliar P(t)** calcula `f(c1), ..., f(cn)`; **selecionar** escolhe os indivíduos que comporão a próxima população; **cruzamento** recombina o material genético de dois indivíduos a partir de um ponto de cruzamento; **mutação** modifica um ou mais genes; e ao final retorna-se a melhor solução encontrada.

```python
from deap import base, creator, tools

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
```

```
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selRoulette)
```

4.2 Seleção pela roleta

O método de seleção de pais deve simular o mecanismo de seleção natural, em que **pais mais capazes geram mais filhos, ao mesmo tempo em que os pais menos aptos também podem gerar descendentes**. Essa segunda cláusula é fundamental: se apenas os melhores se reproduzissem, a diversidade se perderia rapidamente.

No **método da roleta**, cada cromossomo recebe um pedaço proporcional à sua avaliação. Para os indivíduos `0001`, `0011`, `0100` e `0110`, com avaliações 1, 9, 16 e 36 (soma 62), as fatias são 1,61%, 14,51%, 25,81% e 58,07%. Operacionalmente, sorteia-se um número entre 0 e 1 — por exemplo 0,8 — e multiplica-se pela soma: `0,8 * 62 = 49,6`. Percorrem-se as somas acumuladas 1, 10, 26 e 62, selecionando o primeiro indivíduo cuja acumulada seja maior ou igual a 49,6, isto é, `0110`. Com 0,4, tem-se `24,8` e seleciona-se `0100`. O processo se repete até preencher a nova população.

Formalizado como método por computador: (1) encontre a soma da aptidão de todos os membros, `AT = soma(Ai)`; (2) gere um número aleatório `rand` entre 0 e `AT`; (3) pegue o primeiro membro `Ik` cuja aptidão somada às dos precedentes seja maior ou igual a `rand`. O exemplo numérico usa dez cromossomas com aptidões 8, 2, 17, 7, 2, 12, 11, 7, 3 e 7, cujas acumuladas são 8, 10, 27, 34, 36, 48, 59, 66, 69 e 76; para os números aleatórios 23, 49, 76, 13, 1, 27 e 57, selecionam-se os cromossomas 3, 7, 10, 3, 1, 3 e 7 — note que o cromossoma 3, de maior aptidão, é selecionado três vezes.

```
def selecao_roleta(populacao, aptidoes):
    total = sum(aptidoes)
    r = random.uniform(0, total)
    acumulado = 0
    for individuo, apt in zip(populacao, aptidoes):
        acumulado += apt
        if acumulado >= r:
            return individuo
    return populacao[-1]
```

4.3 Operadores e sua escolha dinâmica

O repertório de cruzamentos é ampliado: **1 ponto de corte**, em que os filhos herdam a parte esquerda de um pai e a direita do outro; **2 pontos de corte**, em que o segmento central é trocado — ilustrado com um exemplo aplicado de alocação de poços e turnos; **uniforme**, em que cada gene é sorteado independentemente de um dos pais conforme um padrão binário; e **baseado em maioria**, que usa três ou mais pais, atribuindo a cada gene o valor majoritário. Quanto à mutação, além da troca de um único bit, apresenta-se a **mutação uniforme**, em que cada gene pode ser alterado independentemente segundo uma probabilidade.

Um ponto importante é que os AGs podem incorporar diversos operadores, e surge a pergunta de qual usar a cada instante. A orientação é que **os operadores não devem ser usados todos com a mesma intensidade em cada fase da evolução** — mais crossover no início e mais mutação no final. A solução proposta é usar **uma roleta que sorteia um operador a cada reprodução**, sendo os pesos parâmetros do algoritmo. A intuição é clara: no início a população é diversa e o crossover recombina bem a informação genética presente; no final a população converge e apenas a mutação introduz material novo, evitando estagnação.

4.4 Técnicas e parâmetros

Entre as gerações há dois riscos opostos. É possível que **todos os pais sejam descartados** e seus filhos se tornem os pais da nova geração, perdendo-se bons indivíduos; ou que os indivíduos tenham uma "expectativa de vida" proporcional à sua qualidade, o que faz o tamanho da população crescer caso a avaliação de todos seja muito boa.

As técnicas apresentadas são:

- **Elitismo**: o melhor cromossoma de `P(t)` é copiado em `P(t+1)`, após a mutação e o crossover. Reduz o efeito aleatório do processo seletivo e **garante que o melhor indivíduo da próxima geração seja melhor ou igual ao da anterior**.
- **Steady State**: substituição parcial de indivíduos a cada geração. Bons indivíduos são preservados, ganhando mais chances de reprodução, e os mantidos não precisam ser reavaliados. O método é: crie `n` filhos (seleção, crossover e mutação); elimine os `n` piores membros; avalie e introduza os filhos. O parâmetro **GAP** é a fração da população trocada. No exemplo, uma população com avaliações de 120 a 5 recebe seis novos indivíduos (38, 6, 121, 88, 58 e 17), substitui os piores e, após ordenar, passa a ter como melhor avaliação 121.
- **Steady State sem duplicados**: idem, com exclusão de duplicados, que são mais frequentes com steady state por gerar populações mais estáticas. Garante maior eficiência do paralelismo de busca, assegurando `pop_size` indivíduos diferentes.
- **Ajuste dos parâmetros**: variação dos parâmetros do AG durante a execução, para maior desempenho. Os parâmetros mencionados são taxa de crossover, taxa de mutação, taxa de incremento da normalização da aptidão e pesos dos operadores. A técnica usada é a **interpolação linear**, que define valores inicial e final do parâmetro e a frequência de ajuste.

4.5 Desempenho

O desempenho pode ser medido pelo **grau de evolução alcançado durante o processo evolucionário**. Devido à natureza estocástica dos AGs, é necessário avaliar o **resultado médio de vários experimentos** — uma única execução não é evidência suficiente. O gráfico típico é a curva do melhor indivíduo `f(t)` ao longo das gerações, que cresce acentuadamente nas primeiras gerações e tende a se estabilizar. Os aspectos importantes a observar são a **convergência** do AG, a **proximidade dos melhores cromossomas a um mínimo local** e a **diversidade da população**.

4.6 Exercício aplicado: localização de antenas

O exercício de telecomunicações pede o uso do Evolver para encontrar a resposta ótima de cada etapa, avaliando o desempenho do AG frente a diferentes parâmetros evolutivos. Na **Parte A**, uma empresa deseja otimizar a localização de três antenas, maximizando a cobertura total em relação ao número de clientes atendidos; devem-se encontrar as coordenadas `(x, y)` de cada antena, dados os raios de 15, 12 e 3 km, com dez cidades de coordenadas e clientes conhecidos (por exemplo, cidade 1 em (18, 42) com 7571 clientes e cidade 10 em (50, 46) com 11344 clientes). Na **Parte B**, as antenas já estão em A (22; 11), B (12; 33) e C (41; 37), e busca-se o raio de cada uma que minimize custos garantindo cobertura de todas as cidades, com cada antena atendendo ao menos três cidades, ao custo de R$ 970,00 por km. Na **Parte C**, cada cidade tem percentual de clientes e preço médio distintos, e escolhem-se tanto localização quanto raio, com custos fixos por faixa (de 30 a 45 km, R$ 180.000; de 15 a 30 km, R$ 115.000; de 5 a 15 km, R$ 68.000; de 0 a 5 km, R$ 27.000) somados a R$ 970,00 por km.

A tarefa de Breakout Room pede que, a partir do resultado da Parte A, se alterem os parâmetros do otimizador — **tamanho da população** e **taxa de mutação** — armazenando os resultados e debatendo os achados com os colegas.

5 Tratamento de restrições / Algoritmos Genéticos no Excel

5.1 Tipos de restrições e estratégias

A grande maioria dos problemas envolve restrições, e o material distingue dois tipos: as do tipo **soft**, desejáveis mas que podem ser desobedecidas se necessário; e as do tipo **hard**, que obrigatoriamente devem ser obedecidas. A distinção é operacional: restrições hard tipicamente exigem descarte, reparo ou decodificadores; restrições soft admitem penalização.

O **descarte** simplesmente exclui da população as soluções que não respeitam as restrições. É a estratégia mais simples, mas custosa quando a região viável é pequena.

A **penalização** faz com que as soluções que violam restrições tenham suas avaliações penalizadas, a fim de **manter as características desses indivíduos** na população — essa é sua vantagem central sobre o descarte, pois um indivíduo inviável pode conter material genético valioso. Os tipos de função de penalização, em relação ao grau de violação (desvio), são: **linear**, `Pen(x) = alpha * desvio`; **quadrática**, `Pen(x) = (alpha * desvio)^2`; e **logarítmica**, `Pen(x) = log_n(1 + alpha * desvio)`, onde `alpha` é constante. A escolha define a agressividade: a quadrática pune fortemente desvios grandes, empurrando a busca para a região viável; a logarítmica é branda e preserva indivíduos próximos da fronteira.

```
def avaliacao_penalizada(x, alpha=1.0):
    valor = funcao_objetivo(x)
    desvio = max(0.0, grau_de_violacao(x))
    return valor - (alpha * desvio) ** 2
```

O **reparo da solução** corrige por um algoritmo específico as soluções que violam restrições, exigindo conhecimento do domínio. Os **decodificadores** transformam os cromossomos em soluções válidas, garantindo viabilidade por construção.

5.2 A família Genocop

O **Genocop I** auxilia a otimização considerando somente **restrições lineares**; sua estratégia consiste em diminuir o espaço de busca, tentando encontrar uma solução inicial buscando regiões possíveis. O exemplo é a otimização de `f(x1, ..., x6)` sujeita a restrições como `2*x1 + x2 + x3 = 6`, `x3 + x5 - 3*x6 = 10`, `x1 + 4*x4 = 3`, `x2 + x5 <= 120`, além de limites por variável. A ideia operacional é que, para cada `xk`, existe um intervalo possível quando as outras variáveis são fixas. Assim, para o ponto possível `(x4, x5, x6) = (10, 8, 2)`, tem-se `x4` em `[7.25, 10.375]`, `x5` em `[6, 11]` e `x6` em `[1, 2.666]`. Mutações e ajustes ocorrem apenas dentro desses intervalos, garantindo viabilidade.

O **Genocop II** trata as **restrições não lineares**.

O **Genocop III** incorpora **duas populações separadas**: a **população de busca** (`Ps`), que satisfaz as restrições lineares, e a **população de referência** (`Pr`), que satisfaz todas as restrições. O procedimento é:

```
begin
  P = p                     // replacement probability
  if isfeasible(S) == false
      Z = a*S + (1 - a)*R           // a em [0, 1]
      while isfeasible(Z) == false
          Z = a*Z + (1 - a)*R
      end while
      if evaluation(Z) > evaluation(R)
          R = Z
      end if
      if rand() <= P
          S = Z
      else
          evaluation(S) = evaluation(Z)
      end if
  end if
end
```

A leitura conceitual é elegante: quando um indivíduo `S` da população de busca é inviável, gera-se um ponto `Z` na combinação convexa entre `S` e um indivíduo viável de referência `R`, aproximando-se de `R` até que `Z` seja viável. Se `Z` for melhor que `R`, ele passa a integrar a população de referência. E, com probabilidade `P`, o próprio `S` é substituído por `Z`; caso contrário, `S` permanece na população de busca mas herda a avaliação de `Z`. Assim, o método explora regiões inviáveis sem perder o vínculo com a viabilidade.

5.3 Exercício da fábrica

O enunciado pede a modelagem de um problema de mix de produção: uma empresa produz dois produtos distintos e possui uma fábrica com três setores independentes, com capacidade ociosa de 4, 12 e 18 horas, respectivamente. O lote do produto 1 tem lucro de R$ 3.000,00 e o do produto 2, R$ 5.000,00. A demanda é maior que a capacidade produtiva, e os dois produtos disputam essa capacidade. As condições são: a produção do produto 1 passa pelos setores 1 e 3, enquanto o produto 2 passa pelos setores 2 e 3; a produção não pode consumir mais de 4 horas no setor 1; o produto 2 consome 2 horas do setor 2, que não pode ultrapassar 12 horas; e o produto 1 consome 3

horas do setor 3 enquanto o produto 2 consome 2 horas, sem ultrapassar 18 horas. As restrições de capacidade são hard, e a estratégia natural é penalizar ou reparar soluções que as ultrapassem.

6 Avaliação e Otimização com Múltiplos Objetivos

6.1 Por que múltiplos objetivos

Alguns problemas apresentam múltiplos objetivos simultâneos. O exemplo é o de uma empresa que quer, ao mesmo tempo, entregar todas as quantidades de pedidos de forma correta e pontual; minimizar o número de caminhões que fazem a entrega; diminuir o custo das entregas; e reduzir o tempo despendido. Esses objetivos são tipicamente conflitantes. Os métodos apresentados são: agregação de objetivos; minimização de energia; distância ao alvo; métodos baseados em pesos; separação dos objetivos; conjuntos Pareto; e priorização de objetivos.

A **agregação de objetivos** é um método bastante simples em que, para uma determinada solução, o valor de avaliação final é dado pela **soma ponderada dos valores de avaliação de todos os objetivos**.

Na **distância ao alvo**, as avaliações são combinadas pelo cálculo da distância entre um **vetor alvo** `user`, formado pelos valores ideais de cada objetivo quando considerados isoladamente, e um **vetor** `f`, formado pelos valores de avaliação de todos os objetivos para uma determinada solução. A avaliação final é o somatório das distâncias. O parâmetro `p` determina a pressão exercida sobre as soluções ruins: quanto maior `p`, maior a penalidade aplicada às soluções com resultados ruins em algum objetivo. A limitação é que sua utilização está restrita a situações em que **se sabe previamente a solução desejada para cada objetivo**.

Na **minimização de energia**, durante a otimização os pesos são atualizados de maneira que **pesos maiores são atribuídos aos objetivos menos satisfeitos** pela população; o objetivo do método é minimizar os pesos aplicados aos objetivos. O termo `fnorm_i` é o vetor normalizado de avaliações do objetivo `i`; a atualização dos pesos envolve o termo `e_{i,t}`, que mede o **erro percentual** entre o valor ideal para o objetivo `i` e a avaliação média da população para esse objetivo no instante `t`; as constantes `k1` e `k2` fazem com que a soma dos pesos em um instante arbitrário forneça uma medida do **estado de convergência** do sistema em relação às especificações do usuário. Esse estado corresponde à **energia** do sistema, e sua minimização corresponde ao processo de satisfação de múltiplos objetivos.

6.2 Métodos baseados em pesos

O funcionamento se dá pela aplicação de pesos a cada objetivo: a função de avaliação passa a ser o somatório de todas as avaliações ponderadas pelos respectivos pesos. O exemplo dos slides é instrutivo porque mostra como a escolha dos pesos **inverte a decisão**. Consideram-se dois indivíduos avaliados em três objetivos, com domínios `[0, 10000]`, `[0, 10]` e `[0, 100]`: `x1` com 150; 10; 99 e `x2` com 220; 3; 50.

Com pesos 1, 2 e 1: `F(x1) = 150 + 2*10 + 99 = 269` e `F(x2) = 220 + 2*3 + 50 = 276`, de modo que `x2` é preferido. Já com pesos 0,001, 2 e 0,1: `F(x1) = 0,001*150 + 2*10 + 0,1*99 = 30,05` e `F(x2) = 0,001*220 + 2*3 + 0,1*50 = 11,22`, e agora `x1` é preferido. A lição prática é que, quando os objetivos têm escalas muito diferentes, pesos ingênuos deixam o objetivo de maior

magnitude dominar a decisão; os pesos precisam corrigir a escala além de expressar a preferência do decisor.

O exercício proposto pede avaliar quatro indivíduos sob quatro configurações de pesos. As avaliações são: A com 150, 10 e 99; B com 205, 8 e 40; C com 180, 7 e 88; D com 200, 9 e 50. Os conjuntos de pesos a testar são (1, 1, 1), (1, 2, 1), (1, 1, 3) e (2, 1, 1).

A **separação dos objetivos** consiste em tratar cada função objetivo de forma independente, pegando o máximo obtido em cada objetivo e aplicando ao problema; esse tipo de abordagem é chamado de **otimização não-Pareto**. Operacionalmente, a partir da população formam-se a População A, avaliada pela Função de Avaliação A, e a População B, avaliada pela Função B; as duas populações avaliadas são recombinadas na População Avaliada final.

6.3 Conjuntos Pareto

Estas abordagens **comparam soluções sem combinar avaliações**. O conceito fundamental é a **dominância**: uma solução A domina a solução B se, para nenhum dos objetivos, a avaliação de A é pior que a de B; e, para no mínimo um objetivo, a avaliação de A é melhor que a de B.

O exemplo é um problema de planejamento em que se deseja **minimizar** os custos de produção (`f1`) e de distribuição (`f2`), com as soluções A (2; 10), B (4; 6), C (8; 4), D (9; 5) e E (7; 8). Aplicando a definição, C domina D, pois é melhor em ambos os objetivos; e B domina E, pela mesma razão. A, B e C não se dominam entre si, pois cada uma é melhor em um dos objetivos.

O **Conjunto Pareto-Ótimo** é o conjunto de todas as soluções que não são dominadas por nenhuma outra. A evolução é feita privilegiando os indivíduos desse conjunto, e a avaliação final de uma solução pode levar em consideração o número de indivíduos na população que ela domina — a avaliação **baseada em ranking**.

O exercício pede identificar o conjunto Pareto-ótimo, em um problema de minimização, para os indivíduos A (2; 10), B (4; 6), C (4; 8), D (5; 9), E (8; 7), F (1; 9), G (3; 8), H (6; 1) e I (9; 3).

```python
def domina(a, b):
    """True se a domina b em um problema de minimizacao."""
    return (all(x <= y for x, y in zip(a, b)) and
            any(x < y for x, y in zip(a, b)))

def frente_pareto(solucoes):
    return [s for s in solucoes
            if not any(domina(o, s) for o in solucoes if o != s)]
```

6.4 Implementação em Python e exercício de plantação

O material de apoio inclui um exercício multiobjetivo com a função **DENT**, com dicas de implementação em DEAP: usar `hof = tools.ParetoFront()` como Hall of Fame; registrar `tools.selNSGA2` como método de seleção baseado na frente de Pareto; para a mutação, implementar uma função própria que respeite o domínio ou usar `tools.mutPolynomialBounded` com `low=lb`, `up=ub`, `indpb=0.5` e `eta=0.5`; e criar o gráfico para analisar a frente de Pareto.

```
from deap import tools

hof = tools.ParetoFront()
toolbox.register("select", tools.selNSGA2)
toolbox.register("mutate", tools.mutPolynomialBounded,
                 low=lb, up=ub, indpb=0.5, eta=0.5)
```

O último exercício aplicado reúne restrições e objetivo de lucro: uma área de 1798 hectares será utilizada para o plantio de café, soja, milho, laranja e cana de açúcar, com lucros de R$ 120, R$ 160, R$ 75, R$ 140 e R$ 140, respectivamente. O objetivo é maximizar o lucro total, identificando a melhor distribuição das terras e ilustrando o percentual do terreno de cada produto. As restrições são: laranja e cana de açúcar juntas devem ocupar área mínima de 800 hectares, mas a laranja sozinha não pode ultrapassar 10 hectares; o milho deve ocupar no mínimo 360 hectares e o café, no mínimo 180 hectares; e milho, soja e café juntos não podem ultrapassar 899 hectares. Na versão do exercício da fábrica desta aula, acrescenta-se que o número máximo de lotes de cada produto é igual a 10.

7 Síntese da Disciplina

A disciplina percorre um caminho coerente que vai da pergunta "o que é otimizar?" até a construção de um algoritmo evolucionário completo, capaz de lidar com restrições e com múltiplos objetivos conflitantes. O fio condutor é a tese repetida em quase todas as aulas: **os AGs são flexíveis, mas a qualidade dos resultados depende diretamente da qualidade da modelagem do problema** — representação e decodificação, função de avaliação e operadores genéticos.

A Aula 1 estabelece o vocabulário: as seis entidades da otimização e a analogia com a evolução natural, em que indivíduo é solução, cromossoma é representação, população é conjunto de soluções, gerações são ciclos e meio ambiente é o problema. O exemplo da cabra cega já contém em miniatura todos os ingredientes do AG — busca adaptativa guiada pela avaliação, busca paralela e recombinação de coordenadas promissoras.

A Aula 2 aprofunda representação, decodificação e avaliação. Mostra que a codificação binária é simples e didática mas nem sempre adequada, e que a representação por reais é preferível em domínios contínuos e grandes, por dispensar decodificação, oferecer maior precisão e evitar os Hamming Cliffs. Mais importante, revela que a função de avaliação é objeto de projeto, não um dado: os problemas de **superindivíduos** e de **competição próxima** mostram que uma avaliação numericamente correta pode ser evolutivamente inútil, e as técnicas de **normalização** e **windowing** existem para corrigir isso. O problema das quatro rainhas, com suas três abordagens de codificação, sintetiza a lição sobre representação.

A Aula 3 fecha o ciclo do algoritmo: apresenta o fluxograma completo, detalha o método da roleta em sua versão operacional por soma acumulada, amplia o repertório de operadores e introduz o controle de nível superior — uma roleta de operadores com pesos ajustáveis, mais crossover no início e mais mutação no fim. Elitismo, steady state e steady state sem duplicados resolvem a tensão entre preservar boas soluções e manter diversidade. E o cuidado metodológico final é notável: como os AGs são estocásticos, o desempenho deve ser medido pela média de vários experimentos, observando convergência, proximidade a mínimos locais e diversidade da população.

A Aula 4 confronta o algoritmo com a realidade das restrições, distinguindo soft de hard e oferecendo quatro estratégias: descarte, penalização (linear, quadrática ou logarítmica), reparo e decodificadores. A família Genocop apresenta soluções progressivamente mais sofisticadas — o Genocop I reduzindo o espaço de busca sob restrições lineares, o Genocop II tratando restrições não lineares e o Genocop III mantendo duas populações, uma de busca e outra de referência, com um mecanismo de combinação convexa que traz indivíduos inviáveis de volta à região viável sem destruir seu material genético.

A Aula 5 generaliza a noção de qualidade. Quando há vários objetivos conflitantes, a avaliação deixa de ser um escalar natural. As respostas vão da mais simples à mais sofisticada: agregação e métodos baseados em pesos, que reduzem tudo a um número — com a armadilha das escalas, demonstrada pelo exemplo em que uma mudança de pesos inverte a preferência entre `x1` e `x2`; distância ao alvo, que exige conhecer os valores ideais; minimização de energia, que ajusta pesos dinamicamente favorecendo objetivos mal atendidos; separação de objetivos, a abordagem não-Pareto; e, finalmente, as abordagens baseadas em **dominância de Pareto**, que produzem não uma resposta única, mas um conjunto de compromissos igualmente defensáveis.

As conexões entre as aulas são fortes. A função de avaliação, tema da Aula 2, reaparece na Aula 4 como veículo da penalização por restrições e na Aula 5 como objeto de agregação multiobjetivo. A seleção por roleta da Aula 3 é exatamente o que torna crítica a normalização estudada na Aula 2 — sem roleta proporcional à aptidão, superindivíduos não seriam problema. Os parâmetros discutidos na Aula 3, como tamanho da população e taxa de mutação, são o objeto direto do exercício prático da Aula 4. E o problema das antenas de telecomunicações atravessa três aulas, ganhando complexidade a cada passagem: da localização pura (Parte A) para o dimensionamento de raios sob restrição de custo e cobertura (Parte B) e daí para um problema com receita heterogênea por cidade e custos fixos por faixa de raio (Parte C).

Do ponto de vista prático, a disciplina alterna deliberadamente duas ferramentas: a planilha (Excel, Solver, Evolver), que torna visível o processo de busca e a avaliação, e o Python com DEAP, que permite construir o ciclo evolutivo explicitamente. Essa dupla abordagem reforça a mensagem central do curso: um Algoritmo Genético não é uma caixa-preta que se aplica, mas um modelo que se constrói — e cuja qualidade é a qualidade da modelagem que o sustenta.