Computação Bio-Inspirada

Fabrício Olivetti de França

01 de fevereiro de 2020

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Topics

1. Problemas
2. Desafios adicionais
3. Conceitos Básicos
4. Heurísticas
5. Heurísticas Populacionais

Problemas

Problemas

Os problemas do mundo real costumam serem difíceis de resolver:

- O número de soluções é grande demais para verificar todas as possibilidades
- O problema é difícil de ser formulado computacionalmente, e precisamos simplificá-lo
- A função que avalia a qualidade de uma solução pode ser ruidosa ou variar com o tempo
- Existem muitas restrições associadas

Problemas de Busca e de Otimização

Um problema pode ser formulado como um **problema de busca** ou **problema de otimização**.

Problemas de Busca

Problema de busca: Dado um conjunto de soluções candidatas X e a propriedade $P : X \to \{V, F\}$, encontre um $x \in X$ tal que $P(x)$.

SAT

O problema satisfatibilidade booleana (SAT) é definido como "dada uma expressão booleana $F(x)$, atribuir valores de verdadeiro ou falso para cada variável x_i de tal forma que a expressão avalie para verdadeira.

SAT

Por exemplo:

$F(x) = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3)$

SAT

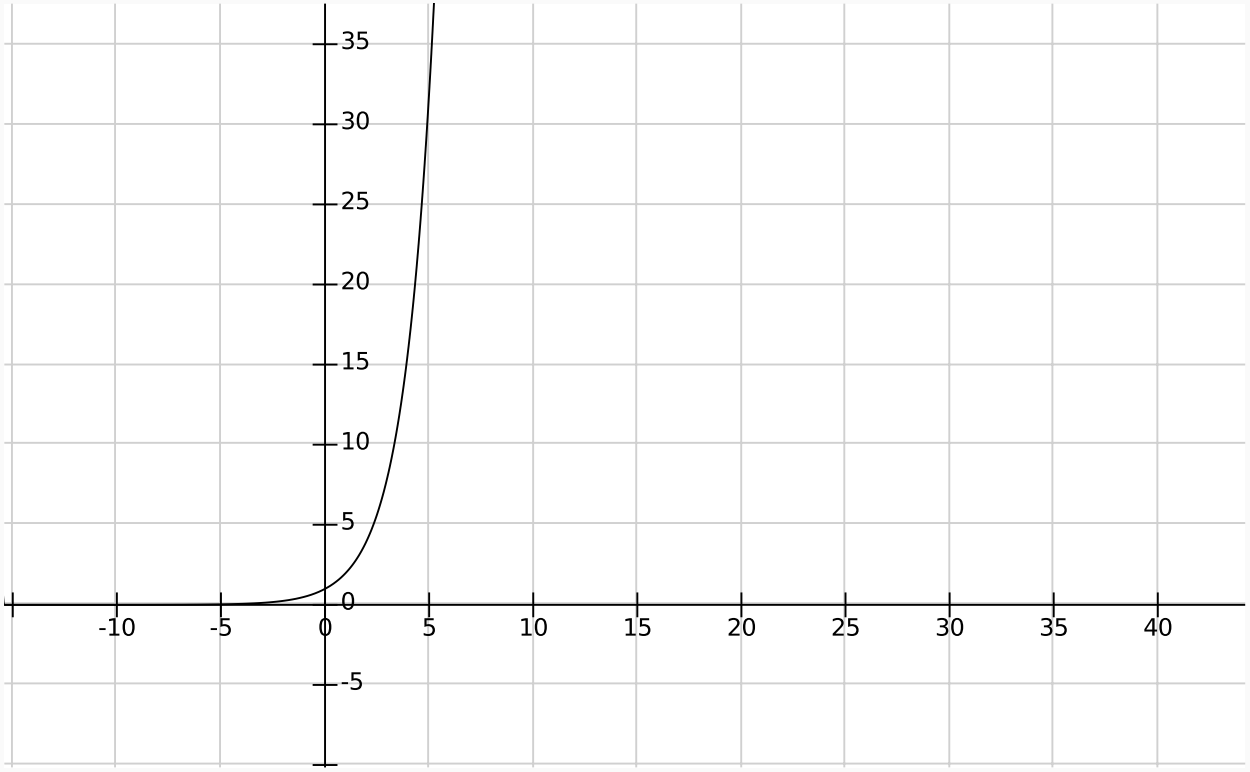
Para facilitar, podemos utilizar 0 e 1 para representar falso e verdadeiro, respectivamente. Temos então, $2^4 = 16$ soluções candidatas:

x_1	x_2	x_3	x_4
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1

...

SAT

Conforme a quantidade de variáveis aumenta, o número de soluções candidatas aumenta exageradamente:



SAT

n	2^n
1	2
5	32
10	1024
100	1.2e30
1000	1.1e301

Problemas de Otimização

Problema de otimização: Dado um conjunto de soluções candidatas X e uma função de critério $F : X \to \mathbb{R}$, encontre um $x \in X$ tal que $x \in \mathit{argmax}_{y \in X} F(y)$.

Otimização não-linear

$$G2(x) = \left| \frac{\sum_{i=1}^{n} cos^4(x_i) - \prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

sujeito a:

$\prod_{i=1}^{n} x_i \geq 0.75$

$\sum_{i=1}^{n} x_i \leq 7.5n$

$0 \leq x_i \leq 10$ for $1 \leq i \leq n$.

Otimização não-linear

Conceitualmente o número de soluções candidatas é infinito. Porém, em um computador temos um limite da precisão de um *float*.

Otimização não-linear

Assumindo que podemos representar até 6 casas decimais, então teremos 10.000.000 valores distintos para cada variável x_i.

Com isso temos $10.000.000^n = 10^{7n}$ soluções candidatas.

Caixeiro Viajante

O Problema do Caixeiro Viajante é um problema de otimização combinatória em que, dado um grafo ponderado com n vértices queremos encontrar um **ciclo euleriano** de menor custo.

Ou seja, qual a sequência de nós que deve ser visitada de tal forma que cada nó seja visitado uma única vez, exceto pelo primeiro que deve ser o final do nosso caminho.

Caixeiro Viajante

Um candidato a solução desse problema é simplesmente uma permutação dos n vértices do grafo descontados das permutações que representam o mesmo ciclo:

1 - 2 - 3 - 4

2 - 3 - 4 - 1

3 - 4 - 1 - 2

...

Caixeiro Viajante

Com isso o conjunto de soluções candidatas possui $n!/(2n) = (n-1)!/2$ elementos.

Problemas de Busca e de Otimização

A diferença entre os dois é que ao encontrar uma solução x para o problema de busca, não precisamos continuar procurando para provar que ela é a solução que queremos. No de otimização é necessário verificar todas as soluções candidatas.

Desafios adicionais

Restrições

Alguns problemas apresentam algumas restrições que removem algumas das soluções candidatas do seu espaço de busca.

Ao remover essas soluções, causa uma descontinuidade no espaço, dificultando a tarefa.

Incertezas

Certos problemas possuem imprecisões na medição da função-objetivo:

- Medições ruidosas
- Custo de medição caro - aproximação
- A medição muda com o tempo
- A solução pode variar ao ser implementada

Múltiplos objetivos conflitantes

Em alguns casos existem mais do que um objetivo a ser atendido, e eles conflitam entre si:

- Segurança x Velocidade de um veículo
- Valor esperado com baixa variância
- Durabilidade x Custo

Conceitos Básicos

Espaço de Busca

O conjunto de soluções candidatas para um problema é denominado **espaço de busca**.

Representação

A escolha da representação da solução de um problema tem um impacto direto no tamanho do espaço de busca e em suas propriedades:

- O espaço é contínuo?
- Duas soluções similares tem uma diferença pequena na função que mede sua qualidade?

Representação

Para o SAT podemos simplesmente utilizar um vetor de números binários.

Representação

No TSP podemos escolher um vetor de inteiros representando a ordem de visita dos nós.

Representação

Para a otimização não-linear, podemos representar como um vetor de números binários ou um vetor de números de ponto flutuante.

Representação

Considere o problema das 8 rainhas: “Em um tabuleiro de xadrez de tamanho 8x8 queremos posicionar oito rainhas de tal forma que nenhuma delas é atacada pelas outras”.

Uma representação natural é uma matriz com valores binários sendo que o valor 1 representa que existe uma rainha naquela coordenada.

Representação

Porém sabemos que não é permitido que duas rainhas estejam posicionadas na mesma linha, com isso reduzimos nosso espaço de busca para vetores unidimensionais em que x_i indica a linha em que uma rainha está alocada na coluna i.

Função-objetivo

A **função-objetivo** é uma função que recebe uma solução e retorna um valor, geralmente real, que mede sua qualidade.

Cada problema deve especificar se queremos maximizar ou minimizar tal função.

Função-objetivo

Para a otimização não-linear, a função-objetivo é a própria função sendo otimizada.

No caso do TSP podemos simplesmente calcular a distância do percurso de cada solução.

Função-objetivo

Já para o SAT, qualquer solução que não for a correta, resultará simplesmente no valor Falso.

Podemos assumir a função na Forma Normal Conjuntiva e contar quantos termos avaliam para verdadeiro, uma vez que o objetivo é que todos os termos sejam verdadeiros.

Vizinhança

O conceito de vizinhança de uma solução s é definido como todas as soluções próximas a s.

A função de vizinhança pode ser definida de duas formas:

- através de uma medida de distância entre duas soluções
- com uma função que projeta uma solução em um conjunto das partes

Vizinhança

Por exemplo, para o problema de otimização não-linear, podemos definir a distância euclidiana ($f : S \times S \to \mathbb{R}$) entre dois pontos:

$d(x_1, x_2) = \sqrt{\sum_i (x_{1i} - x_{2i})^2}$

E a vizinhança como:

$N(x) = \{y \in S \mid d(x, y) \leq \epsilon\}$

Vizinhança

Para o SAT podemos definir a distância de Hamming que conta quantos bits diferem entre duas soluções.

Vizinhança

A outra forma é definir uma função $f : S \rightarrow 2^S$ que, dado uma solução, retorna um conjunto de soluções vizinhas.

No TSP podemos definir uma função de vizinhança que troca a posição de dois nós na solução:

$[1, 2, 3, 4, 5]$

$[4, 2, 3, 1, 5]$

Vizinhança

Note que a função $f : S \to 2^S$ é equivalente a $f : S \to S \to 2$ e $f : (S \times S) \to 0, 1$.

Ou seja, é uma função que pega duas soluções s_1, s_2 e retorna verdadeiro se elas forem vizinhas entre si.

Busca Local

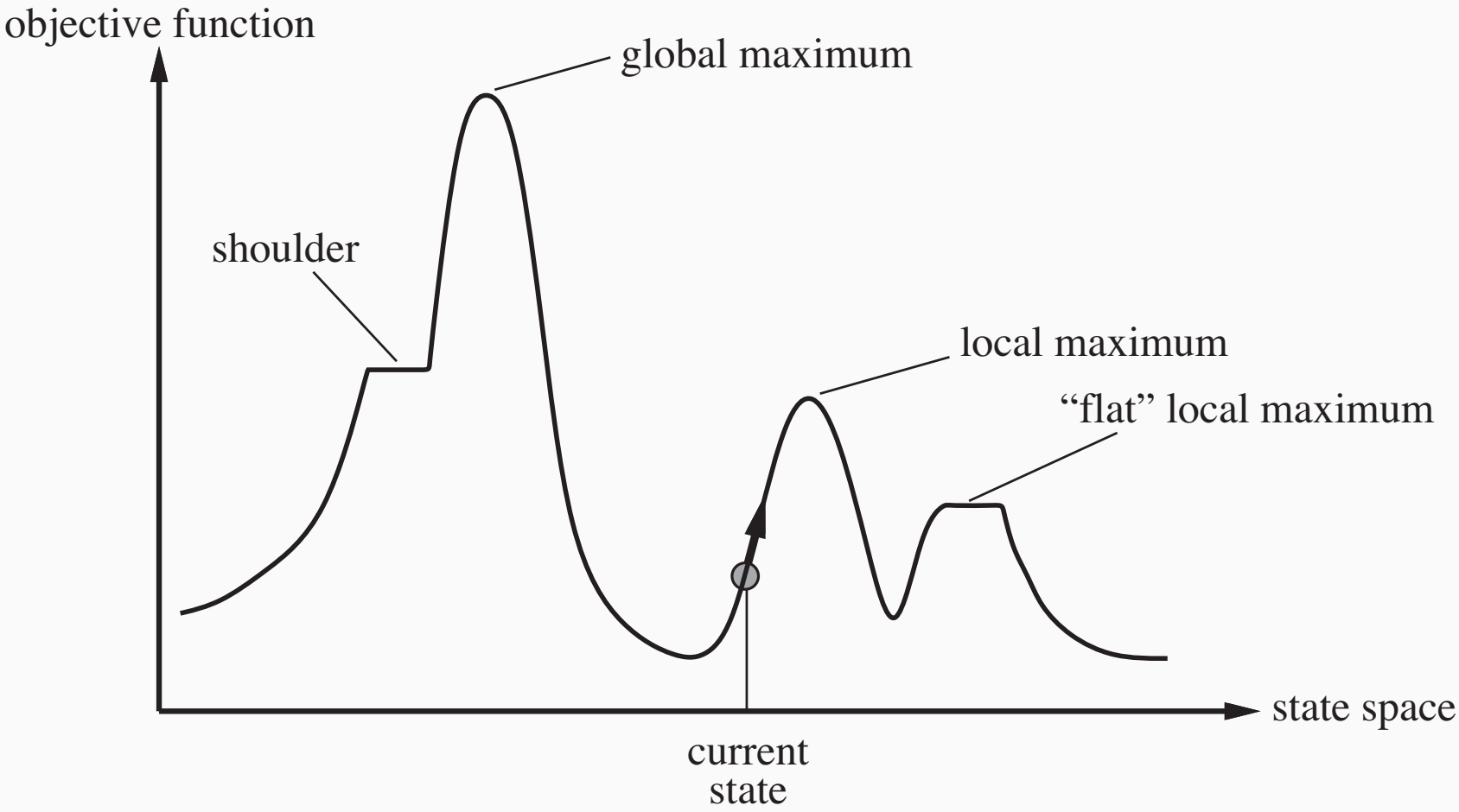
A ideia de vizinhança nos permite pensar em uma busca que partimos de um ponto e prosseguimos, iterativamente, para as soluções vizinhas de melhor qualidade.

Busca Local

Em um problema como o de maximizar a função $f(x) = -x^2$, não importa o ponto que partimos, sempre chegaremos a melhor solução por esse procedimento.

Busca Local

Porém, dependendo da quantidade de ótimos locais, esse procedimento fica *preso* no ótimo mais próximo. Por isso é denominado de **busca local**.



Busca Local

A **Busca Local** parte de uma solução inicial qualquer e, iterativamente, caminha para soluções vizinhas até um ponto de convergência.

Busca Local

Na Busca Local, os passos para chegar até a solução tipicamente não são guardados em memória, ou seja, apenas tem por objetivo encontrar uma solução sem saber os passos que levaram até ela.

Busca Local

Um algoritmo de busca local é dito **completo** se sempre encontra uma solução factível (se existir); e é dito **ótimo** se sempre encontra o melhor dentre essas soluções.

Hill-climbing

O algoritmo de busca local **hill-climbing** ou de **maior subida** simplesmente repete iterativamente os passos:

- Avalia todoso os vizinhos do estado atual
- Caminha para o estado vizinho de maior valor, se maior que o atual

E para quando não existem mais vizinhos melhores.

Hill-climbing

A ideia geral do algoritmo é o de escalar uma montanha pelo lado mais íngrime, ou seja, de subida mais rápida.

Hill-climbing para SAT

Vamos definir o problema SAT para

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

Com solução inicial:

$$x = F, F, F, F$$

$$f(x) = F$$

Hill-climbing para SAT

Vamos definir o custo da solução como a quantidade de termos que avalia como verdadeiro, nesse exemplo temos $c(x) = 2$.

Hill-climbing para SAT

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

Os vizinhos possíveis dessa solução representam a mudança do valor de uma variável:

$$x = F, F, F, V$$

$$x = F, F, V, F$$

$$x = F, V, F, F$$

$$x = V, F, F, F$$

Hill-climbing para SAT

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

Escolhemos um dos melhores e repetimos:

$$x = F, V, F, V$$

$$x = F, V, V, F$$

$$x = V, V, F, F$$

Hill-climbing Estocástico

Escolhe aleatoriamente um dos vizinhos dentre aqueles melhores que o estado atual.

A escolha pode ser proporcional a quanto melhora.

Hill-climbing com Reinício Aleatório

Aplica o Hill-climbing em diferentes estados iniciais e retorna a melhor solução obtida.

Essa estratégia consegue resolver o problema de 3 milhões de rainhas em menos de um minuto!

Busca aleatória

Considere agora um algoritmo que gera uma nova solução aleatória em cada passo.

Ela é uma busca local completa? Ela é ótima?

Essa busca é conhecida como **caminhante aleatório**.

Recozimento Simulado

O algoritmo de Hill-climbing é incompleto pois, dado um estado inicial, está limitado a vizinhança de seu ótimo local.

Recozimento Simulado

Por outro lado, um algoritmo de **caminhante aleatório** é completo pois pode chegar em qualquer estado do espaço de estados, porém é ineficiente.

Recozimento Simulado

Um meio termo é definido pelo algoritmo de **Recozimento Simulado** (**Simulated Annealing**).

Recozimento Simulado

A ideia é que a escolha do próximo estado seja aleatória, se esse estado for melhor que o atual, é aceito, caso contrário ele substitui o atual com probabilidade $e^{\frac{\Delta E}{T}}$, sendo ΔE a diferença entre a função-objetivo desse estado com o estado atual e uma T a *temperatura* que é reduzida a cada iteração.

Recozimento Simulado

O comportamento inicial do algoritmo é de ser mais permissivo quanto a piora do estado mas, com o passar das iterações, ele tende a aceitar apenas estados que apresentam melhoras.

Se a temperatura for diminuída devagar o suficiente, esse algoritmo encontra o ótimo global com probabilidade se aproximando de 1.

Recozimento Simulado

```
1  def SimulatedAnnealing():
2      s0 = solucaoInicial()
3      T = 100
4      while T > eps:
5          s = estadoVizinhoAleatorio(s0)
6          if f(s) > f(s0):
7              s0 = s
8          else:
9              if random() <= exp((f(s)-f(s0))/T):
10                 s0 = s
11         T = reduz(T)
```

Factibilidade x Descontinuidade

O problema proposto pode apresentar restrições que tornam algumas regiões do espaço de busca infactíveis.

Nesse caso o problema pode se tornar mais complicado pois a vizinhança de uma certa solução pode não levar ao ótimo.

Heurísticas

Heurísticas

Heurística, derivada do grego “encontrar” ou “descobrir”, são técnicas para encontrar a solução de um problema sem garantia de obter algum ótimo (nem mesmo local) ou que seja racional.

Heurística

Objetivo de obter um alvo imediato (ex.: colocar a rainha da coluna i na fileira de menor número de ataques).

Heurística

George Pólya enumera algumas dicas para criar uma heurística e tentar resolver um problema (*How to Solve it, 1945*):

- Se não consegue entender o problema, desenhe uma imagem ou diagrama representativo.
- Se não consegue chegar do estado inicial até uma solução, tente partir da solução e chegar ao estado inicial.
- Se o problema é abstrato, crie um exemplo concreto.
- Tente resolver um problema mais genérico primeiro (possivelmente menos restritivo).

Heurística

Métodos heurísticos tem a vantagem de serem adaptáveis para problemas que não temos um ambiente totalmente observável, apresente aleatoriedades, com espaço de estados muito grande e sem objetivo definido (dada uma métrica de qualidade).

Heurística Construtiva

Uma **heurística construtiva** ou **gulosa** (*Greedy heuristic*) é aquela que constrói uma solução iterativamente.

Para ser possível isso, precisamos de uma forma de avaliar uma solução parcial.

Heurística Construtiva

Um exemplo simples é para o TSP em que começamos com um nó inicial e iterativamente adicionamos o nó vizinho mais próximo até completarmos o caminho.

Heurística Construtiva

Para o SAT, iterativamente atribuímos um valor para cada varíavel que maximiza o número de termos verdadeiros.

Heurística Construtiva

Finalmente, para a otimização não-linear podemos fixar os valores de todas as variáveis, exceto x_i e variar o valor de x_i até atingir o ótimo.

Repetimos o processo para as outras variáveis.

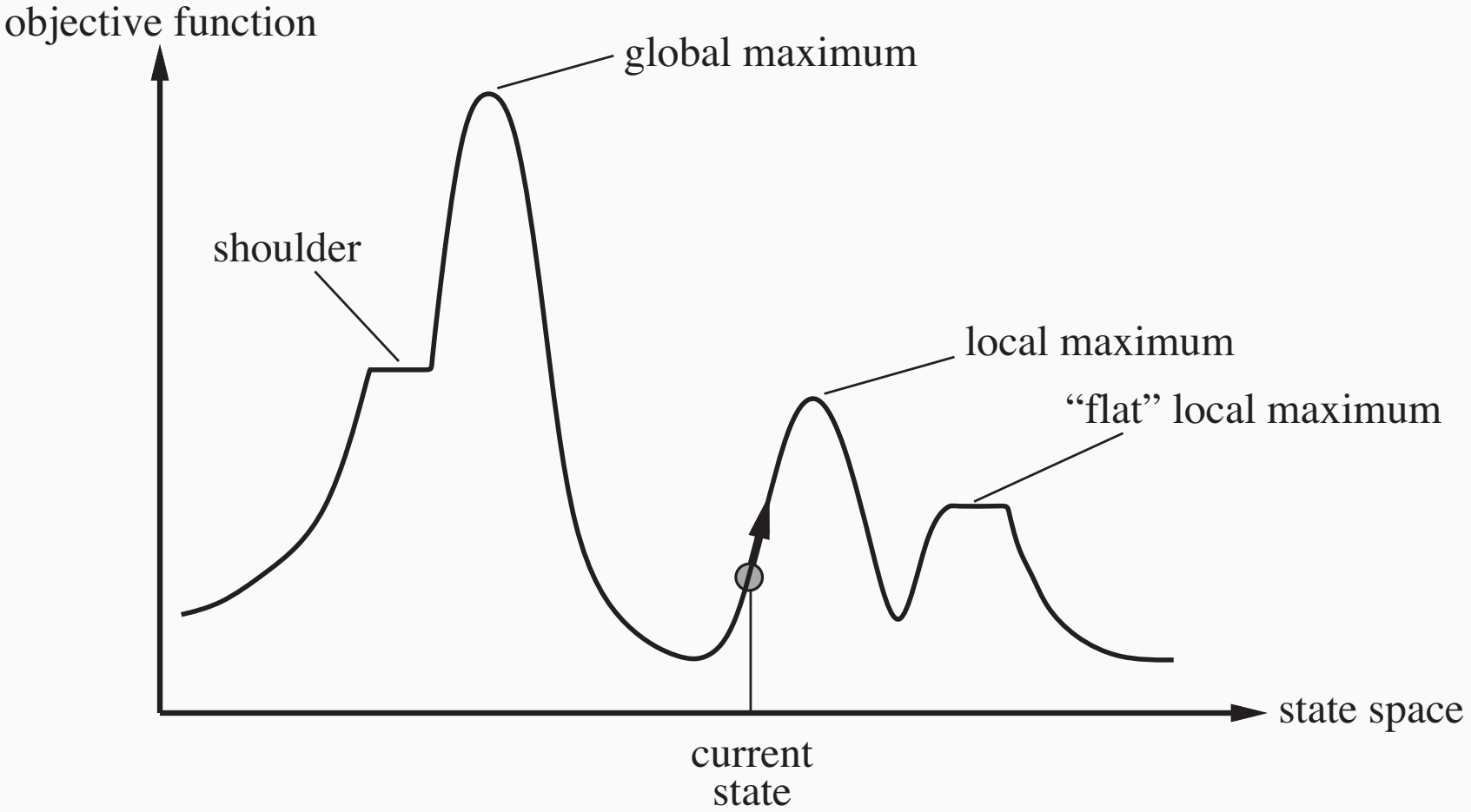
Heurística Construtiva

As heurísticas construtivas geralmente tem um desempenho ruim mas podem ser utilizadas para gerar uma solução inicial para a busca local.

Heurísticas Populacionais

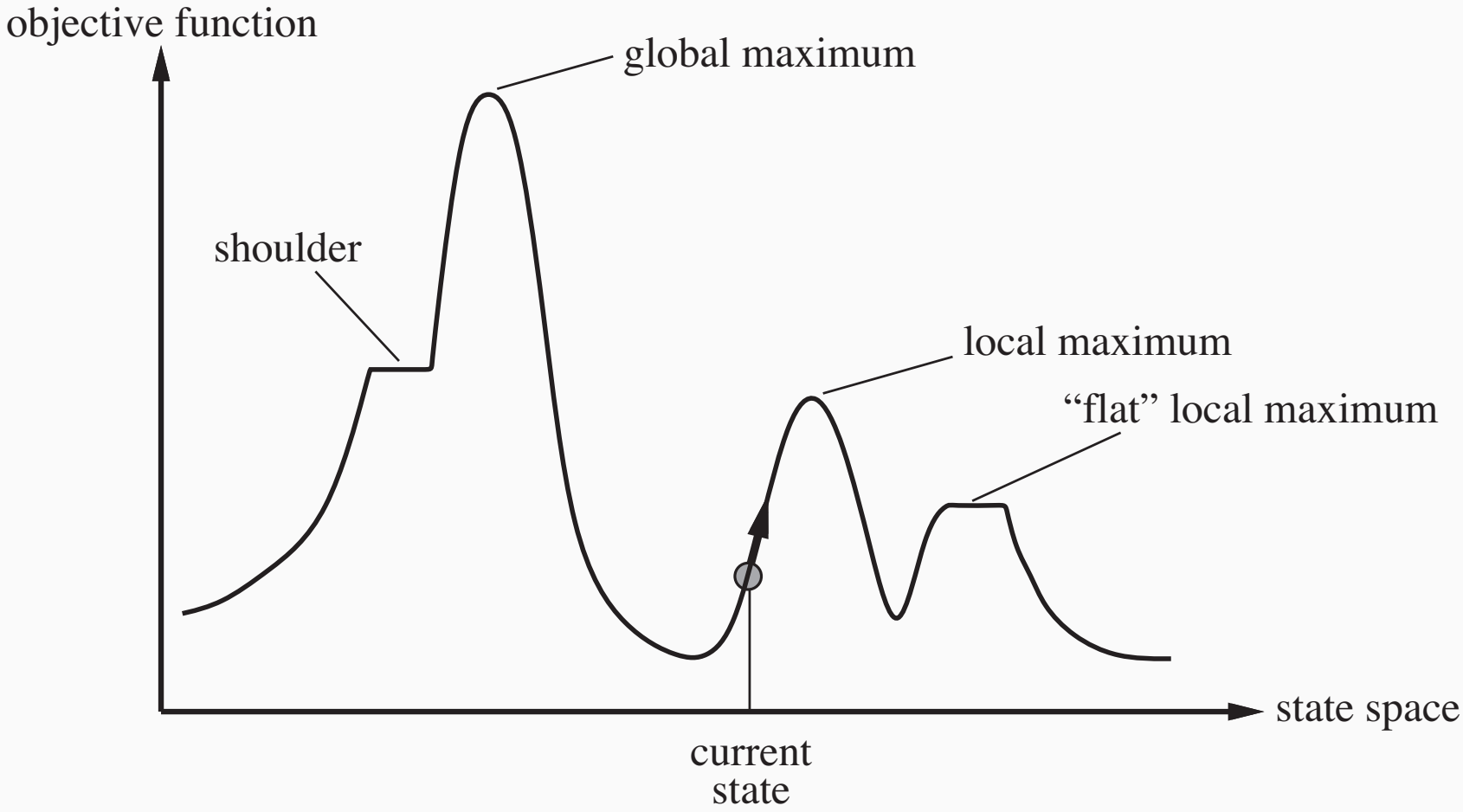
Heurísticas Populacionais

Até então consideramos heurísticas que trabalham com apenas uma única solução atual que é atualizada para o próximo passo do algoritmo.



Heurísticas Populacionais

Mas e se, ao invés de considerarmos apenas uma solução atual, considerarmos várias soluções?



Além da vantagem de permitir a inicialização em múltiplas bases de atração, também podemos executar o procedimento em paralelo.

Mas não é somente isso que podemos ganhar com essa abordagem...

Por exemplo, podemos pensar em criar uma competição entre as diversas soluções estimulando o foco entre as regiões mais promissoras.

Heurísticas Populacionais

Outra ideia é que a vizinhança de uma solução possa ser compartilhada com outras criando combinações de soluções parciais que possuem boa qualidade segundo a função-objetivo.

Heurísticas Populacionais

A ideia principal é que iremos trabalhar com um equilíbrio entre **exploração** e **explotação**.

Exploração x Explotação

Exploração: o ato de explorar toda a região do espaço de busca.

Explotação: o ato de explorar apenas a região de vizinhança.

Heurísticas Populacionais

Digamos que geramos 30 soluções aleatórias para o SAT, podemos fazer isso sorteando cada bit com 50% de chance para cada valor.

Em seguida, selecionamos 30 soluções dessas 30 para fazer uma pressão seletiva. A ideia é que os mais aptos apareçam repetidos na população e os piores desapareçam.

Heurísticas Populacionais

Dentre os selecionados podemos fazer uma variação aleatória e repetir o processo.

Para uma expressão com 91 termos e 20 variáveis, em 30 execuções com 200 soluções e um máximo de 1000 iterações:

- Atinge o objetivo em 24 execuções
- A geração média que atinge o objetivo é 272 iterações

Heurísticas Populacionais

Um algoritmo abstrato para essa ideia pode ser descrito como:

```
pop <- populacaoAleatoria
f   <- avalia pop
enquanto não terminar do
  pop <- seleciona pop
  pop <- altera pop
  f   <- avalia pop
```

Vamos tornar esse algoritmo menos abstrato nas próximas aulas.