INF623

2024/1

UFV

Inteligência Artificial

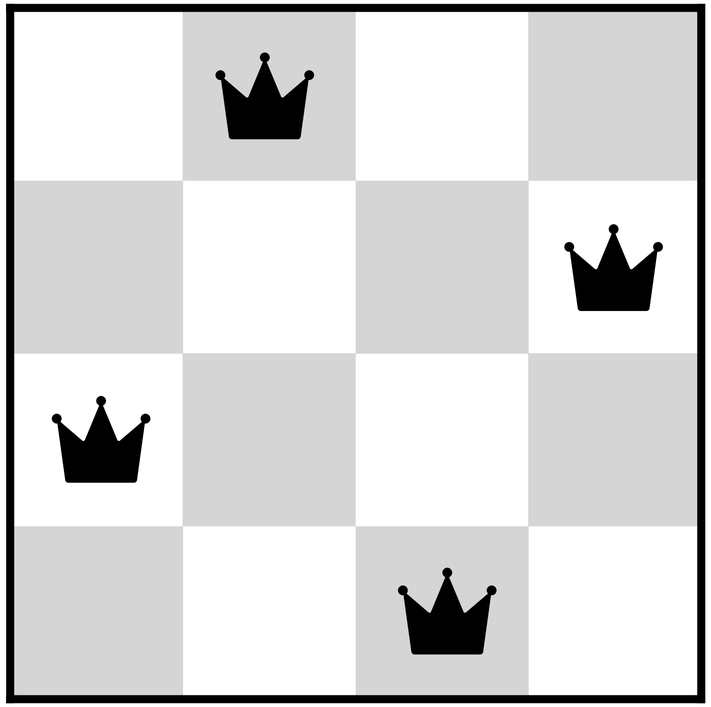
A4: Busca local e otimização I

Plano de aula

- Problemas de otimização
- Espaço de busca
- Algoritmos de busca local
  - Subida de encosta
    - Como evitar máximos (ou mínimos) locais
  - Têmpera simulada
  - Busca em feixe (beam search)

Exemplo 1: nove rainhas

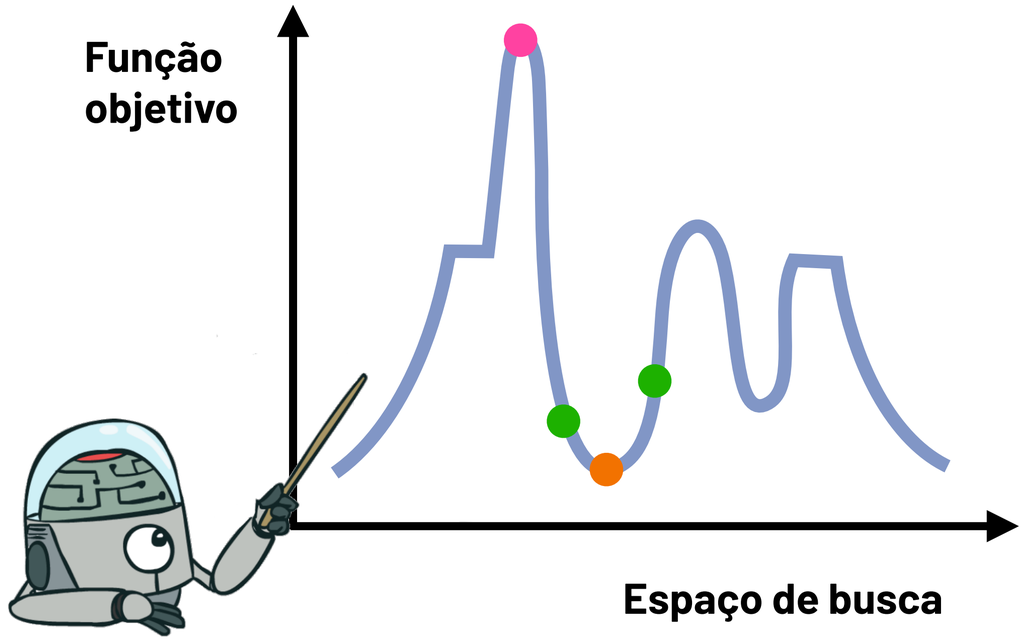
Considere o problema das N-rainhas: posicionar N rainhas no tabuleiro de xadrez de tal forma que elas não se ataquem. Por exemplo, para N=4:



Nesse tipo de problema, nós estamos interessados no estado final em si, e não no caminho para chegar até lá.

Agentes racionais de busca local

Para resolver problemas desse tipo, chamados de **problema de otimização**, um agente assume que o mundo é representado por um **espaço de busca** e que objetivo é encontrar uma **solução que maximiza** (ou minimiza) uma **função objetivo** a partir de um **solução inicial.**



- Em problema de otimização, não temos um conjunto de ações para um determinado estado
- A busca é conduzida explorando a **vizinhança** das soluções.

Espaço de busca

Problemas de otimização

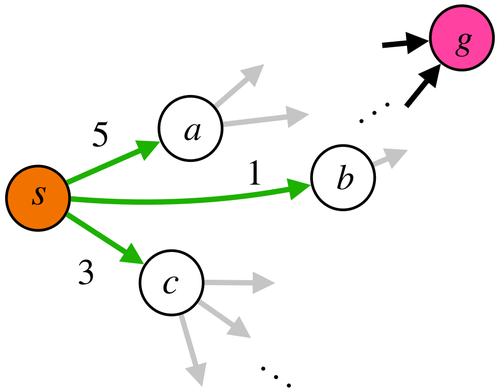
Um **problema de otimização** pode ser definito pela tupla (C, S, b, opt), onde:

- C é uma coleção de candidatos
- $S \subseteq C$ é um conjunto de soluções (que respeitam as restrições do mesmo)
  - Quando o dominio dos candidatos em C é discreto e S é finito chamamos o problema de **otimização combinatória**
- $opt \in \{\min, \max\}$ é um tipo de objetivo
- v é uma função objetivo $v : S \to \mathbb{R}$

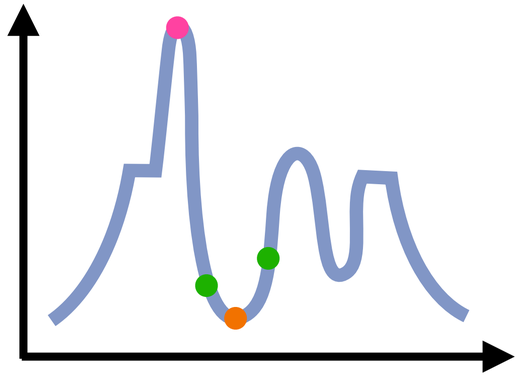
Seja (C, S, b, opt) um problema de otimização, a solução ótima v^* é definida como:

$$v^* = \begin{cases} min_{c \in C}\ v(c) \text{ se } opt = \min \\ max_{c \in C}\ v(c) \text{ se } opt = \max \end{cases}$$

Busca informada vs. Busca local



Definição de vizinhança é crítico para o desempenho de algoritmos de busca local!



- Objetivo é encontrar um caminho entre s e g
- Busca no espaço de **estados**
- **Função de ações A:** dado um estado, gerar próximos estados
- h estima a distância para uma solução e deve ser definida na modelagem

- Objetivo é encontrar g (o caminho não importa)
- Busca no espaço de soluções candidatas
- **Vizinhança:** dado um candidato, gerar candidatos vizinhos
- h é dada pela função objetivo v

Exemplo 1: problema das N-rainhas como busca local

- **Candidatos:** cada rainha ocupa uma coluna do tabuleiro
- **Função heurística h:** número de rainhas que se atacam (minimização)
- **Vizinhança:** modificar a posição de uma rainha em sua coluna
  - Por exemplo, um dos vizinhos do candidato da Figura 1 consiste na rainha da terceira coluna uma casa para cima;
  - Note que essa estratégia, por definição, não permite a geração de um vizinho que tenha duas rainhas em uma mesma coluna

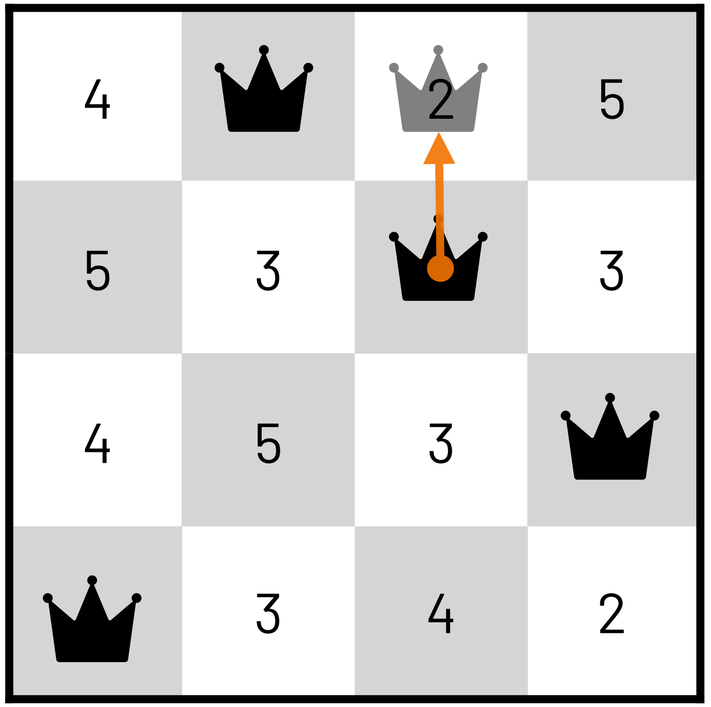


Figura 1: 4 rainhas se atacam; os números mostram o nu´mero de rainhas que se atacam.

Algoritmo da subida de encosta (SDE)

Assumindo um problema de maximização (opt = max)

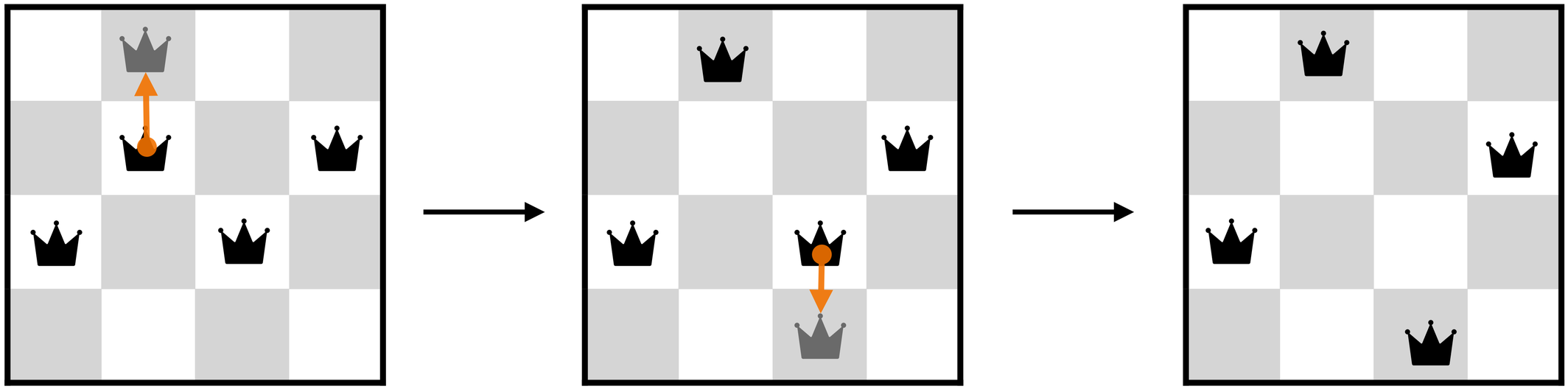
```
def SDE(C, h):
 1. atual = candidato aleatório em C
 2. while True:
 3.     prox = vizinho de atual com maior valor h
 4.     if h(prox) <= h(atual):
 5.         return atual
 6.     atual = prox
```

Propriedades:

- Complexidade de memória constante
- Sempre termina se o conjunto de candidatos é finito
- **Não é completo:** sem garantia de que o resultado é uma solução
- Se o resultado é uma solução s, entra s é um ótimo local (em relação a h)

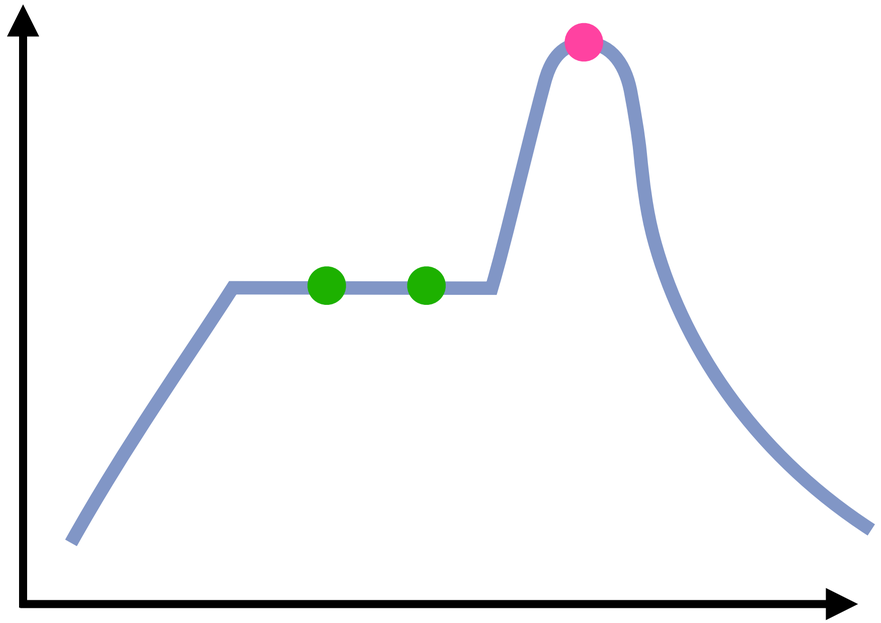
Subida de encosta no problema das rainhas

Exemplo de execução do SDE começando de um candidato com **h = 5** e uma sequência de **dois movimentos** que geram uma solução **(h = 0)**.



h = 5 **h = 2** **h = 0**

Como evitar máximos locais?

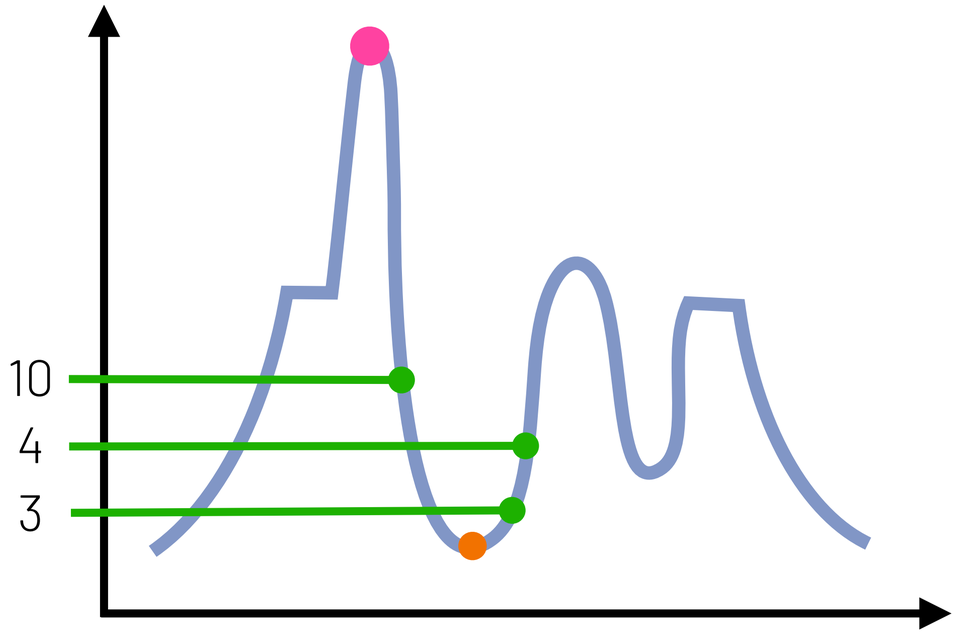


Movimentos laterais

Permite escolher **vizinhos com o mesmo valor de heurística** h(**prox**) == h(**atual**)

- Pode causar repetições infinitas em máximos locais planos
- Limita-se o número de movimentos laterais

Como evitar máximos locais?

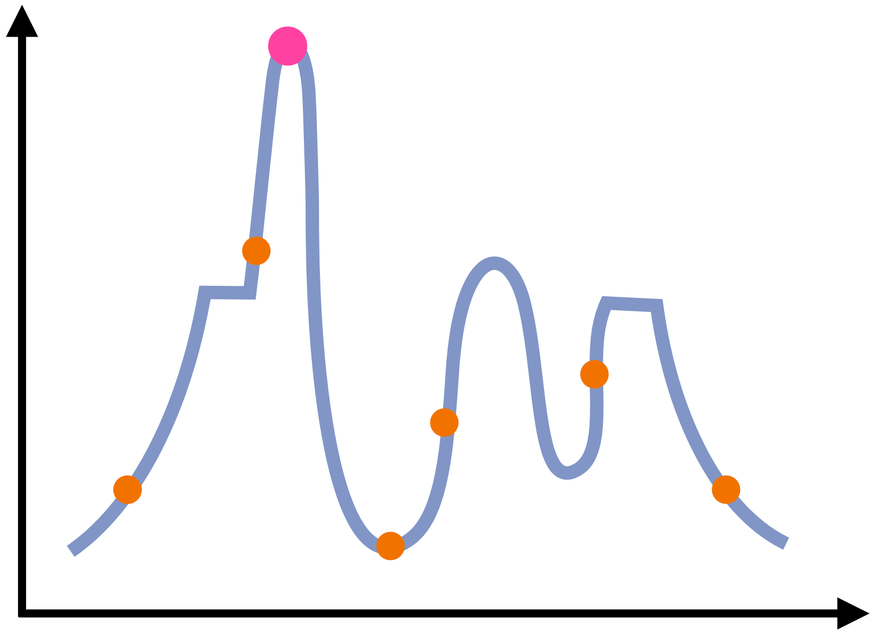


Subida de encosta estocástica

Escolhe-se o vizinho **`prox`** aleatoriamente com probabilidade proporcional ao valor de **`h(prox)`**

- Em um problema de maximização, candidatos **`prox`** com maior h(**prox**) são escolhidos mais frequentemente
- Para candidatos com $h(3,4,10)$, teríamos as probabilidades aproximadas (0.17, 0.23, 0.58)
- Converge mais lentamente, mas pode encontrar soluções melhores

Como evitar máximos locais?



Reinício aleatório

Executa-se a subida de encosta várias vezes, sempre inicializando de um local aleatório

- Estratégia completa e ótima
- Quando o número de reinício tende a infinito, a probabilidade de encontrar uma solução ótima converge para 1

Como evitar máximos locais?

Passeio aleatório

- Inicializa-se o candidato inicial aleatoriamente
- Em casa passo seleciona-se o vizinho prox aleatoriamente
  - Não utiliza função heurística para guiar a busca
  - Estratégia completa e ótima: quando o número de reinício tende a infinito, a probabilidade de encontrar uma solução ótima converge para 1

Algoritmo da têmpera simulada (TS)

Têmpera Simulada (TS) combina subida da encosta com passeio aleatório

Ideia – escolher vizinhos ruins de vez em quando para tentar evitar máximos locais:

- Se **`prox`** é melhor que atual, vá para **`prox`**
- Se **`prox`** é pior que atual, vá para **`prox`** com probabilidade:

$$p = e^{\frac{h(prox) - h(atual)}{T}},$$

onde T é um parâmetro reduzido ao longo das iterações segunda um dado escalonador.

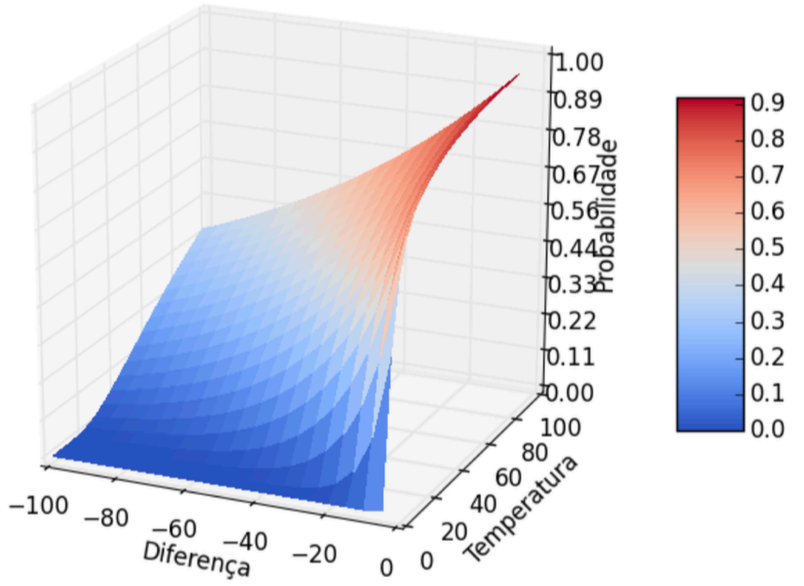
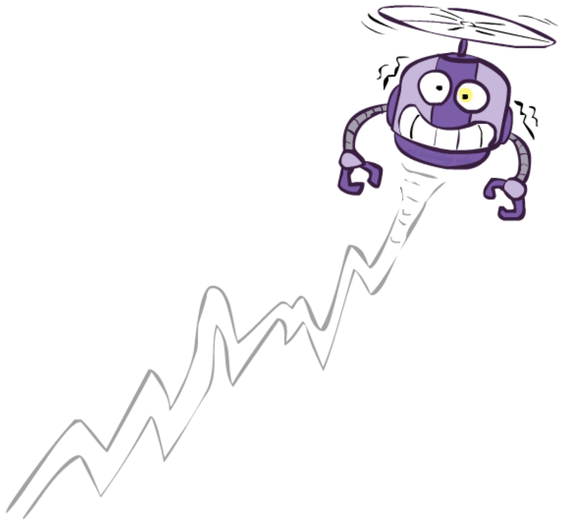


Figura 2: Função de temperatura para vários valores de h(prox) - h(atual) e T

Algoritmo da têmpera simulada (TS)

Assumindo um problema de maximização (opt = max)

```
def TS(C, v, h):
 1. atual = candidato aleatório em C
 2. while True:
 3.     T = valor de acordo com o escalonamento (diminuindo T)
 4.     if T == 0:
 5.         return atual
 6.     prox = vizinho de atual com maior valor h
 7.     if h(prox) > h(atual):
 8.         atual = prox
 9.     else:
10.         com probabilidade p = e^((h(prox) - h(atual))/T), atual = prox
```



Busca em feixe (beam seach)

Similar a subida de encosta, mas ao invés de manter 1 candidato na memória, mantém k (feixe):

- Se $k = 1$, se comporta como subida de encosta
- Se k é muito grande, se comporta como busca em largura

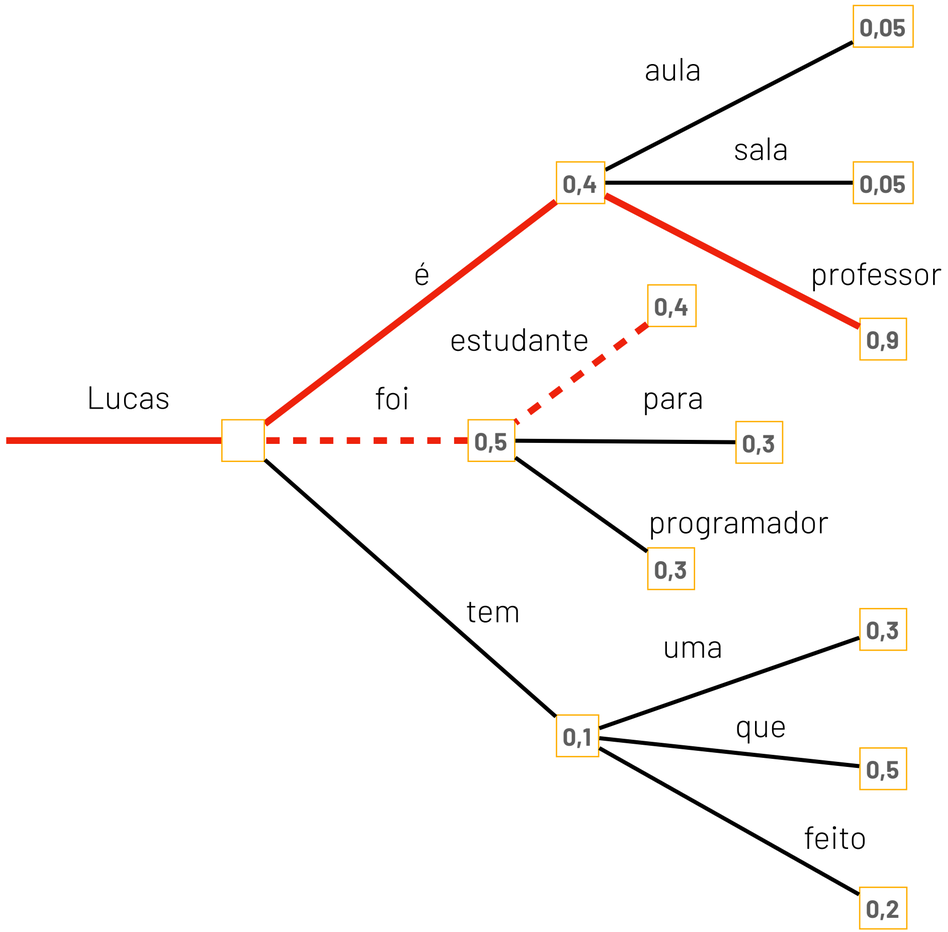


Figura 3: Beam search é muito utilizada atualmente para tradução automática de texto.

Próxima aula

A5: Busca local e otimização II

Algoritmos genéticos: representação, função de adaptação, cruzamento e mutação