Inteligência Artificial

Prof. Doutor Felipe Grando

felipegrando@uffs.edu.br

Introdução a área de IA

O que é IA, fundamentos e breve histórico
(RUSSELL; NORVIG, 2022. cap. 1)

Introdução

- O campo da Inteligência Artificial (IA) visa a construção de máquinas inteligentes que conseguem computar e agir racionalmente de modo eficaz e seguro em uma grande variedade de situações
- Abrange diversos subcampos, tais como:
  - Busca e otimização
  - Aprendizado de máquina
  - Robótica e automação
  - Visão computacional
  - Processamento de linguagem natural

O que é?

- Inteligência
  - Racionalidade, fazer a coisa “certa”
  - Compreender e resolver problemas
  - Aprender com a experiência
  - Adaptar-se a novas situações
  - Pensar no que é correto e porquê
  - Agir de forma eficiente para atingir um objetivo
- Artificial
  - Criado pelo Ser Humano
  - Oposto de natural
  - Mecânico ou tecnológico
  - Inorgânico

Inteligência Artificial

- Subcampo da Ciência da Computação mas é multidisciplinar
- Sistemas de computador que imitam ou buscam superar (singularidade) a inteligência humana
- Sistemas que são capazes de pensar, agir e resolver problemas ou atuar em atividades que normalmente exigiriam inteligência humana
- Agentes racionais – modelo padrão
  - Racionalidade limitada – agir de forma apropriada com recursos limitados de tempo ao invés de buscar a racionalidade perfeita
- Máquinas benéficas e úteis para os humanos

Fundamentos

- Filosofia
  - Discorre que a mente é semelhante a uma máquina no sentido que codifica o conhecimento e opera (pensa) sobre ele para agir inteligentemente
- Matemática
  - Base e ferramenta para compreensão da computação e do raciocínio sobre algoritmos
- Economia
  - Formalização do conceito de utilidade na tomada de decisões
- Engenharia da Computação
  - Construção de máquinas mais poderosas e mais “usáveis” (engenharia de *software*)
- Neurociência
  - Funcionamento do cérebro humano e como ele se assemelha e se diferencia dos computadores
- Psicologia
  - Consideram animais como máquinas de processamento de informações e que o uso da linguagem se ajusta a esse modelo
- Controle e Automação
  - Trouxeram ideias para técnicas e máquinas que usam o *feedback* do ambiente para agirem de forma ótima

Histórico

- 1943 – 1956
  - Redes de neurônios artificiais (1943)
  - Aprendizado hebbiano (1949)
  - Algoritmo genético, aprendizado por reforço e Teste de Turing (1950)
  - Cunhado o termo AI (*Artificial Intelligence*) (1956)
- 1952 – 1969
  - Entusiasmo inicial e grandes expectativas
  - Venceu o melhor jogador de Damas (1956)
  - Criação do Lisp (1958), linguagem de programação dominante para a IA por cerca de 30 anos
  - Provador de teoremas (1959)
  - Perceptron e sua teoria de convergência (1962)
- 1966 – 1986
  - Decepção com resultados previstos para a área
  - Poder computacional muito limitado para técnicas de aprendizado
  - Sistemas especialistas (1969)
  - Criação do Prolog (1972), linguagem de programação com paradigma de programação em lógica matemática

Histórico

- 1986 – Atualidade
  - Aprendizado por retropropagação – *backpropagation* (1986)
  - Criação do Python (1991), ganhou maior popularidade a partir da versão 3 (2008)
  - Foco no raciocínio probabilístico e aprendizado de máquina
  - Evolução significativa no poder computacional
  - Resultados importantes em jogos como o Xadrez (1997), Jeopardy (2011), Go (2015), Dota 2 (2018), pôquer (2019), Starcraft II (2019) e Quake III (2019)
  - Big data (2001) e *World Wide Web* (2007) proporcionaram dados abundantes para técnicas de aprendizado
  - Aprendizado profundo e redes convolucionais (1995)
  - Desenvolvimento e uso de *hardware* especializado e GPUs
  - Fama no reconhecimento da fala e de objetos (2011)
  - Grande avanço na robótica em geral e veículos autônomos

Principais veículos científicos

- Organizações
  - AAAI (American Association for AI)
  - ACM SIGAI (Special Interest Group in AI)
  - European Association for AI
  - Society for AI and Simulation of Behavior
- Revistas (periódicos)
  - Artificial Intelligence
  - Computational Intelligence
  - IEEE Transactions on Pattern Analysis and Machine Learning
  - IEEE Intelligent Systems
  - Journal of AI Research
- Eventos (conferências)
  - IJCAI (International Joint Conference on AI)
  - AAAI Conference
  - ECAI (European Conference on AI)
  - BRACIS (Brazilian Conference on Intelligent Systems)

Agentes Inteligentes

Sistemas que percebem seu ambiente e atuam de forma adequada

(RUSSELL; NORVIG, 2022. cap. 2)

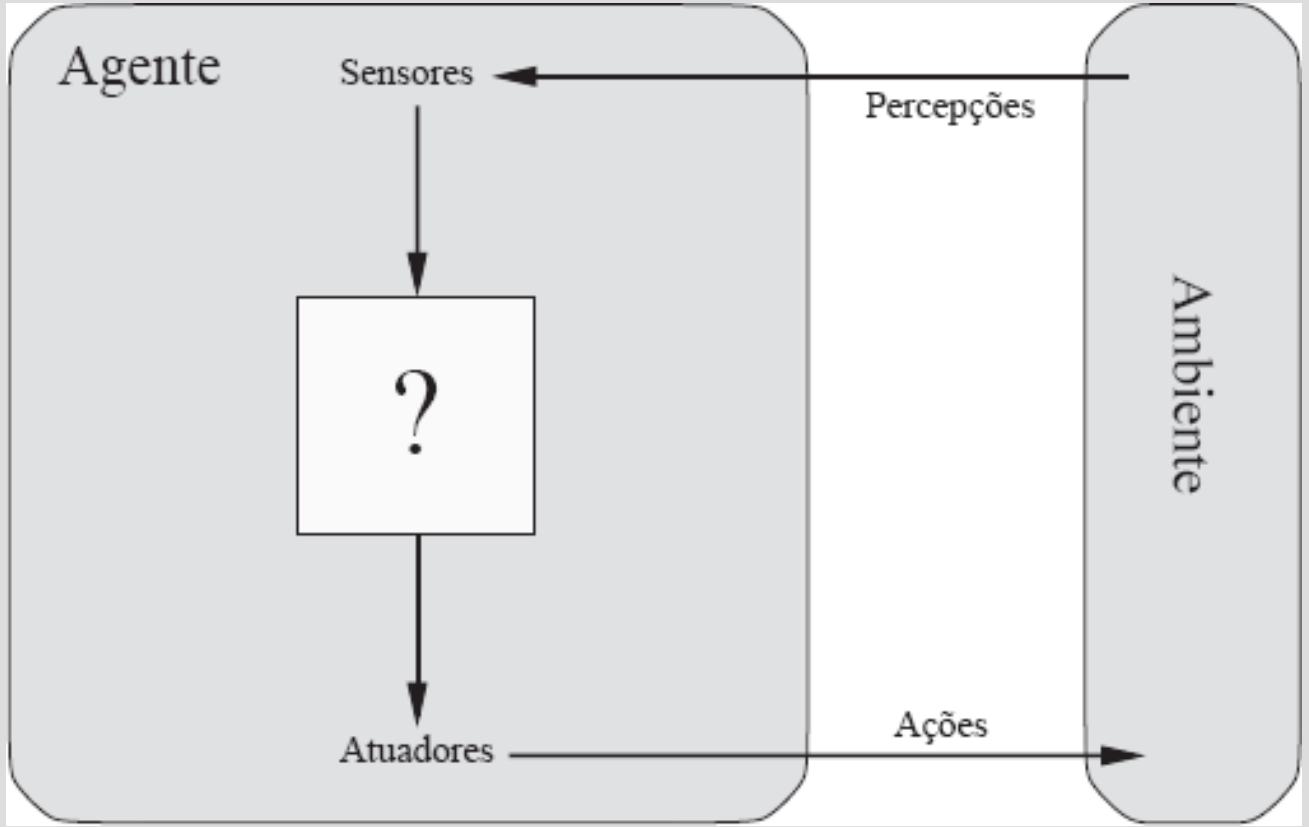
Agentes e Ambientes

A função de um agente é a de mapear um conjunto de percepções do ambiente em uma ação

Um agente é considerado inteligente ou racional se a ação selecionada maximiza sua medida de desempenho

Coletar informações do ambiente (exploração) e armazenar percepções passadas (memorização) pode ser importante para melhorar o desempenho do agente ao modificar sua função de mapeamento (aprendizado)

Um agente é autônomo quando baseia suas ações em suas próprias percepções ao invés de apenas no conhecimento anterior definido pelo seu projetista



Agentes interagem com ambientes por meio de sensores e atuadores

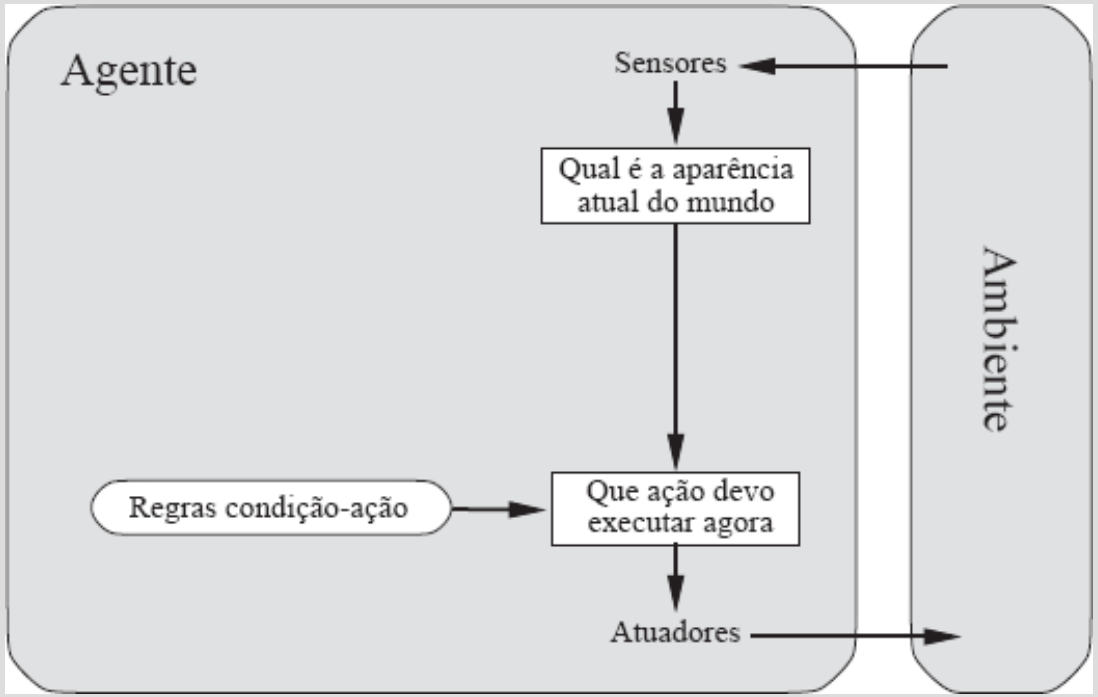
Natureza dos ambientes

- Completamente observável
- Agente único

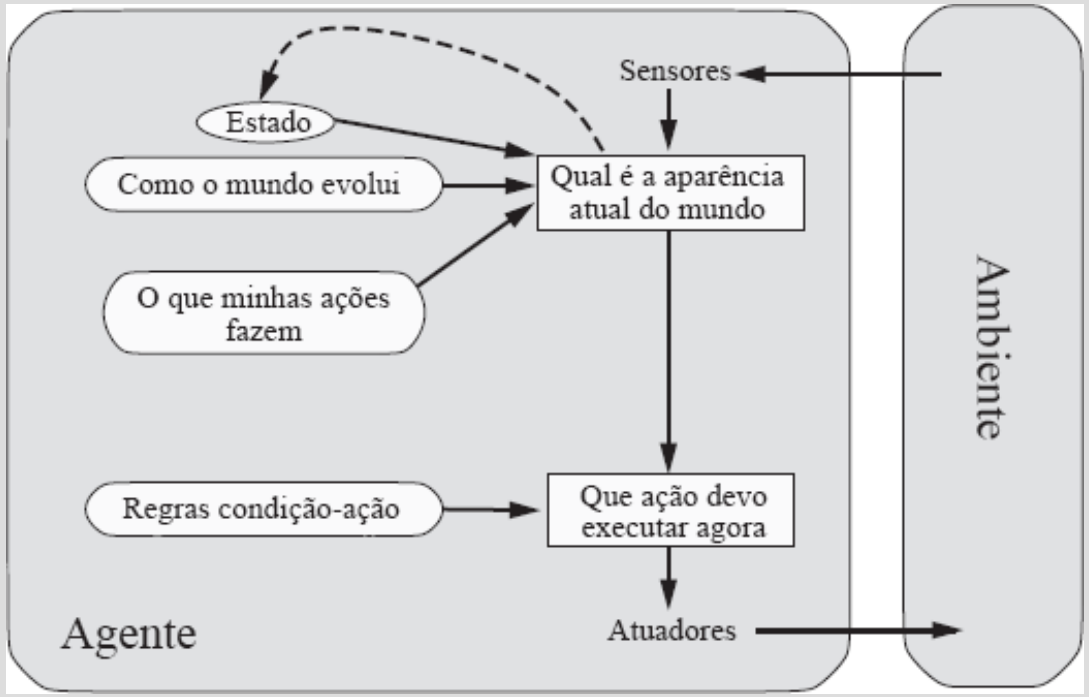
- Determinístico
- Episódico
- Estático
- Discreto
- Conhecido

- Parcialmente observável
- Multiagente
  - Cooperativo ou competitivo
- Não determinístico
- Sequencial
- Dinâmico
- Contínuo
- Desconhecido

Tipos de agentes

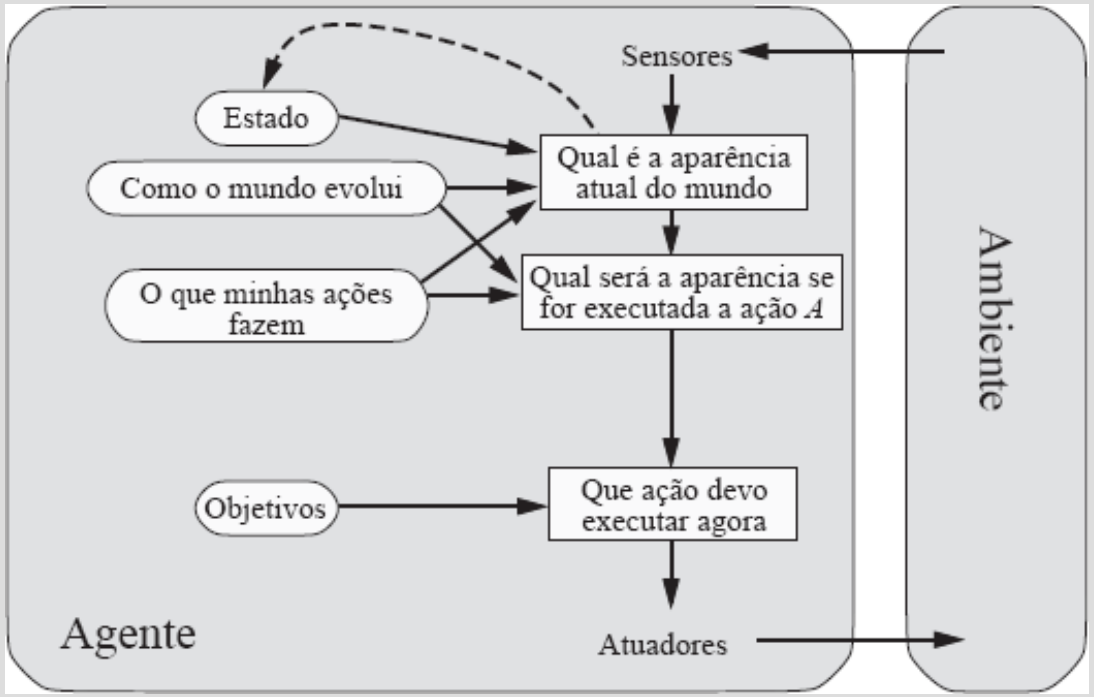


Agente reativo simples

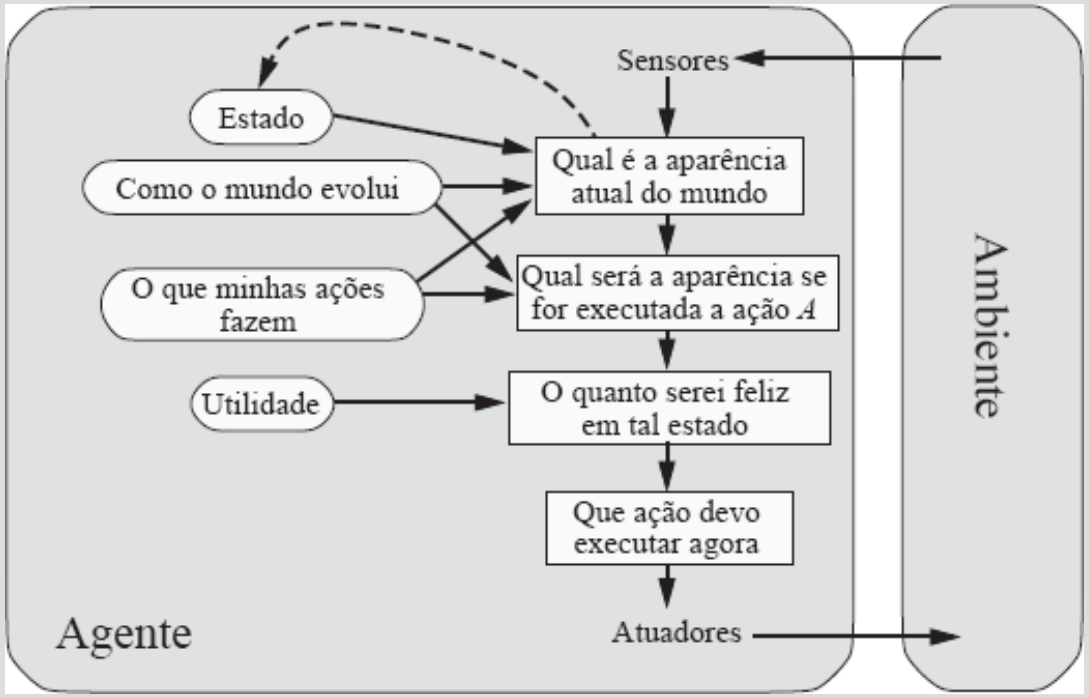


Agente reativo baseado em modelo

Tipos de agentes



Agente baseado em modelo e em objetivos



Agente baseado em modelo e em utilidade

Tipos de agente

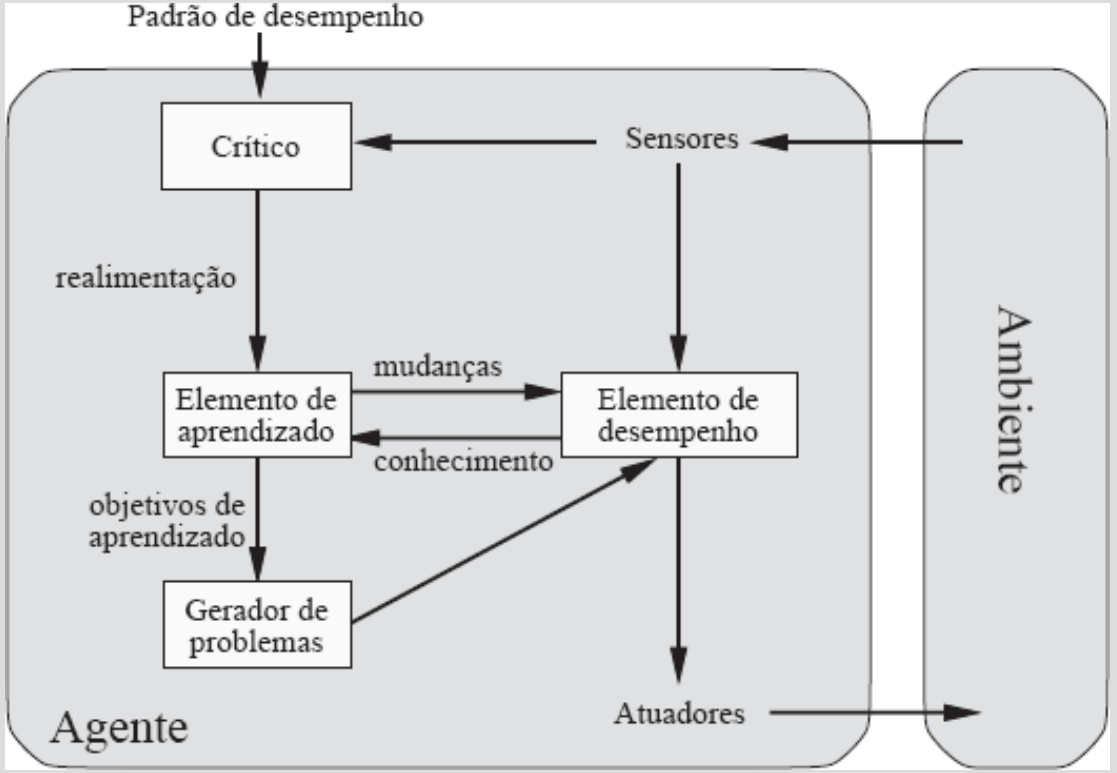
Agentes reativos simples respondem diretamente às suas percepções

Agentes reativos baseados em modelos mantêm o estado interno para monitorar aspectos do mundo que não estão evidentes na percepção atual

Agentes baseados em objetivos agem para maximizá-los ou alcançá-los

Agentes baseados em utilidade buscam maximizar o seu próprio “bem-estar”

Todos os agentes podem melhorar seu desempenho por meio do aprendizado



Agente com aprendizado

Problemas de busca e otimização

O objetivo é encontrar uma sequência de ações para uma solução

(RUSSELL; NORVIG, 2022. cap. 3)

Problemas de busca

- Problemas de busca podem ser definidos como:
  - Conjunto de estados possíveis para o ambiente do problema
    - Cada estado representa uma situação única do ambiente em determinado tempo
    - O conjunto de todos os estados é chamado de espaço de estados
      - O espaço de estados do jogo de Xadrez é estimado em um valor maior ao de átomos no Universo conhecido
  - Estado inicial
    - Pode ser distinto para diferentes instâncias do problema
  - Conjunto de estados finais, meta ou objetivo
    - O conjunto pode ser vazio se a instância do problema não possuir solução
  - Conjunto de ações possíveis
  - Modelo de transição de estados
    - Mapeia ou descreve o que cada ação faz
  - Função de custo ou de objetivo
    - Presente apenas em problemas de otimização. Nesses problemas não é suficiente encontrar uma solução qualquer pois desejamos encontrar a melhor solução possível (solução ótima)

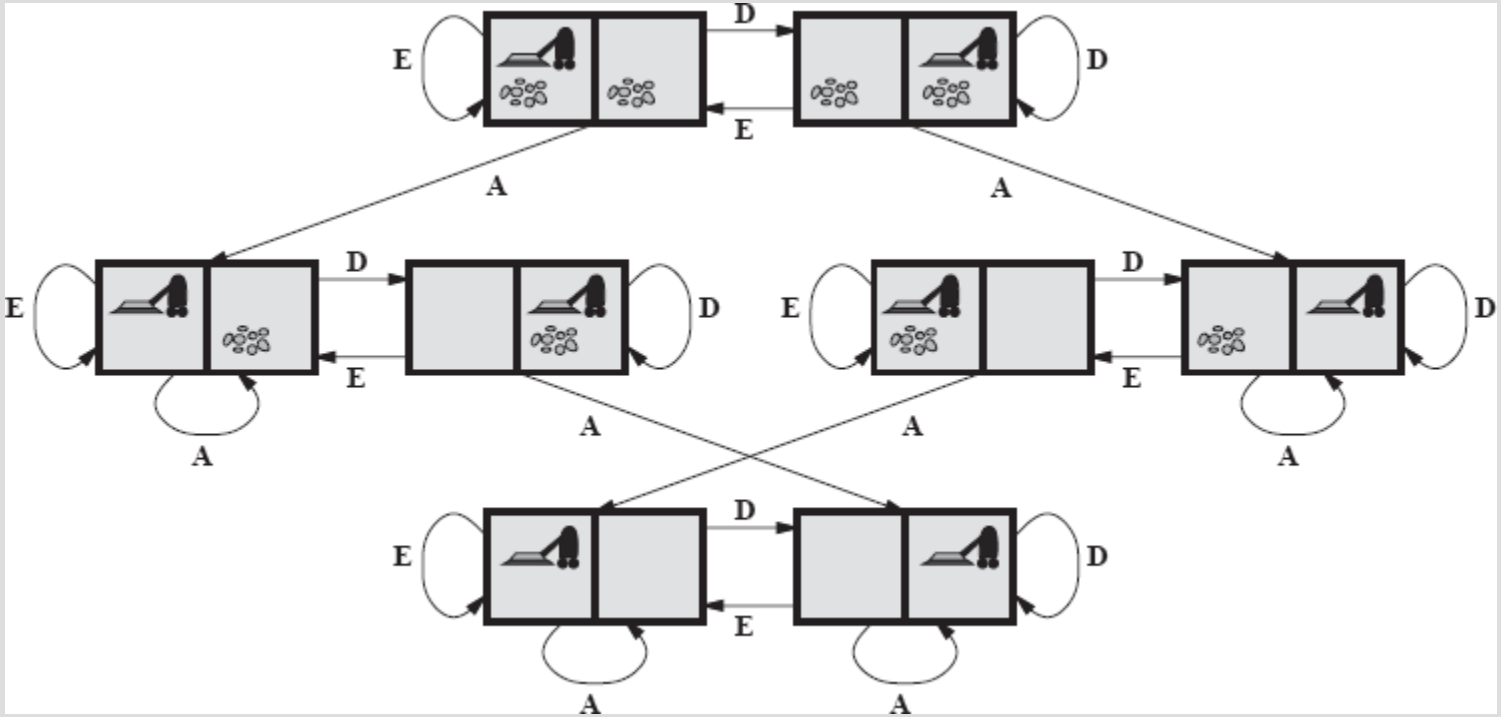
Modelo de um problema de busca

- Um modelo de um problema é uma representação abstrata do problema real com apenas as características consideradas essenciais para a solução adequada do problema
  - O modelo sempre é uma simplificação do mundo real ao qual se deseja atuar e é fundamental para tratar de problemas muito complexos ou grandes
  - Quanto maior o nível de abstração do problema maior é a simplificação do mesmo. O desejável é sempre manter o problema o mais abstrato possível sem perder nenhuma característica importante para a solução do mesmo

Exemplo – Modelo do problema do aspirador

- Estados: 8 possíveis
  - Formado por duas células físicas de espaço que podem estar limpas ou sujas e duas posições para o aspirador
- Estado inicial: qualquer estado pode ser o inicial
- Ações: 3 possíveis
  - Mover o aspirador para direita ou para a esquerda e aspirar a sujeira no local onde se encontra
- Estado final: os estados onde todos os espaços estão limpos
- Custo de ação: cada ação custa 1 unidade de tempo

Exemplo – Modelo do problema do aspirador



Modelo de transição e espaço de estados

Exemplo – Modelo do Jogo de 8 peças

- Estados: 9!/2 = 181440
  - Formado por uma grade 3x3 e 8 peças (1 espaço em branco, sem peça)
- Estado inicial: qualquer estado pode ser o inicial
- Ações: 4 possíveis
  - Mover para cima, baixo, esquerda e direita
- Custo de ação: cada ação custa 1 unidade de tempo
- Modelo de transição: o espaço vazio troca de lugar com uma peça ortogonalmente adjacente ou permanece no mesmo lugar caso a ação seja inválida (mover uma peça para fora da grade)

Exemplo – Modelo do Jogo de 8 peças

- Estado final:
  - Blocos ordenados em ordem crescente com o espaço vazio no canto inferior direito

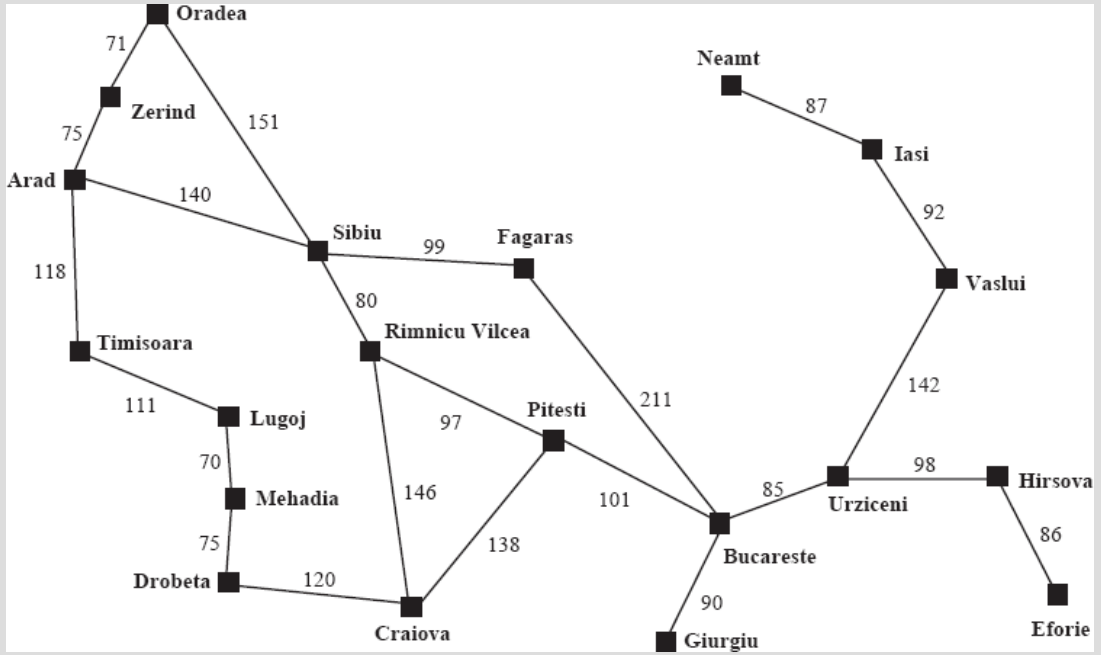
1	2	3
4	5	6
7	8	

Exemplo – Modelo do Caixeiro Viajante

- Estados: V!/V
  - Formado por um grafo com V vértices onde cada vértice representa um local a ser visitado e o vértice inicial deve ser também o último vértice (ciclo hamiltoniano)
    - Podemos abstrair e retirar a repetição do último vértice na sequência pois é uma características inata do problema que a rota deve terminar sempre no vértice inicial
  - Não consideraremos uma rota parcial como sendo um estado possível pois não é uma solução viável
  - Podemos abstrair a necessidade de visitar um mesmo vértice múltiplas vezes em problemas com rotas restritas (grafos incompletos) se computarmos e considerarmos somente a melhor rota entre cada um dos vértices independentemente se essa rota passar por outros vértices intermediários
- Estado inicial: qualquer sequência com todos os vértices do grafo sem repetição
- Ações: mudar a ordem da sequência dos vértices
- Custo de ação: a ação custa tempo computacional variado
- Modelo de transição: uma sequência (lista de vértices) é transformada em outra sequência válida

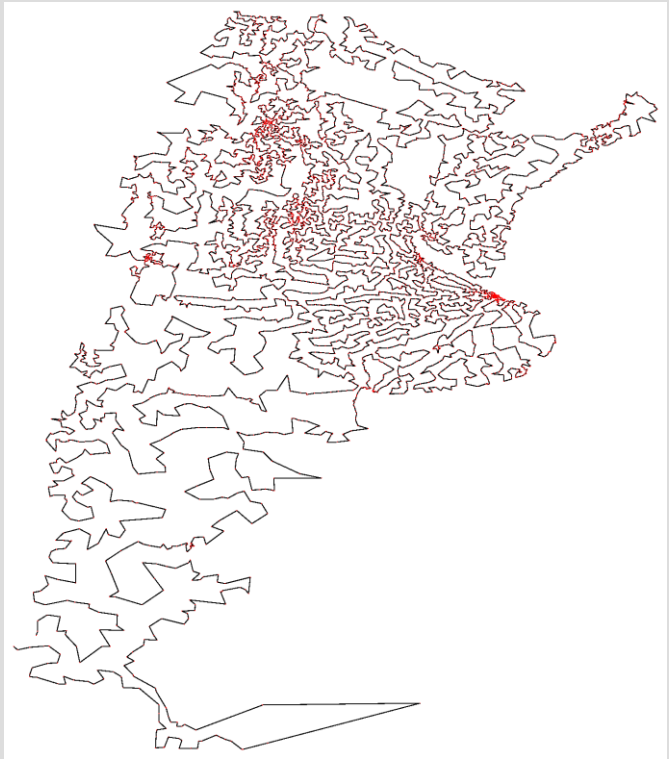
Exemplo – Modelo do Caixeiro Viajante

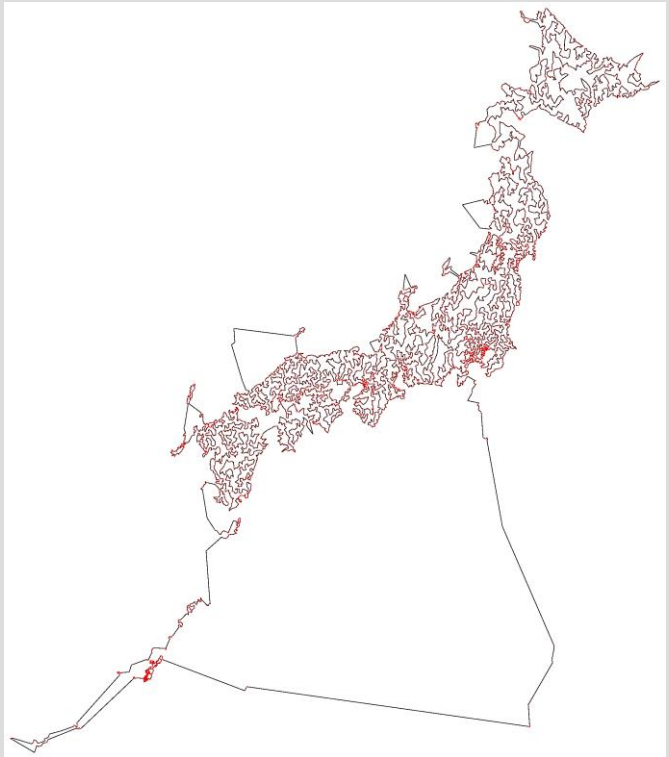
- Exemplo de grafo representando rotas entre cidades da Romênia
  - Cada cidade é representada por um vértice do grafo
  - Cada aresta representa uma rota entre as cidades e o seu peso é o custo dessa rota
    - O custo pode ser a distância, o tempo de viagem ou até uma combinação de múltiplos fatores



Exemplo – Modelo do Caixeiro Viajante

- Uma solução possível para o problema do caixeiro viajante para duas instâncias do mundo real (6723 cidades da Argentina à esquerda e 9847 cidades do Japão à direita)
  - Apenas a solução para a instância do Japão é ótima
  - Foram consideradas apenas as distâncias em linha reta e desconsideradas qualquer característica geográfica





- Instâncias retiradas de:
  - https://www.math.uwaterloo.ca/tsp/world/countries.html

Algoritmos de busca simples

Agentes inteligentes que buscam encontrar uma solução de forma eficiente

(RUSSELL; NORVIG, 2022. cap. 3)

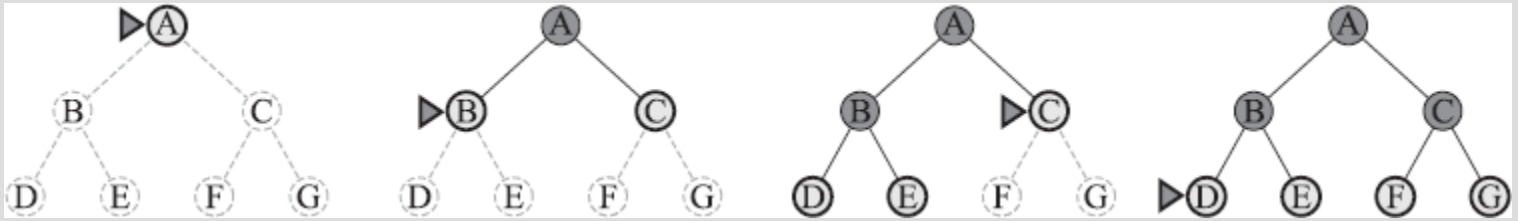
Conceito

- Algoritmos de busca recebem uma entrada (instância) de um problema de busca e visam retornar uma sequência de passos (ações) para um estado meta (solução)
- Muitos algoritmos de busca sobrepõem uma árvore de busca sobre o espaço de estados, formando diferentes caminhos a partir do estado inicial (raiz) até o estado meta (nó folha)
  - As conexões ou ramificações na árvore de busca, entre os nós pais e filhos, são geradas pelas ações possíveis em cada estado pai
  - A busca realiza a expansão (análise) dos nós na árvore de busca até encontrar uma solução
    - Cada algoritmo de busca aplica uma estratégia diferente de escolha do nó a ser expandido primeiro
- Podemos avaliar uma busca quanto a sua completude (encontra uma solução, se existir uma), custo e/ou eficiência (tempo e espaço gastos) e otimalidade (sempre encontra a melhor solução, ou seja, aquela cujo somatório do custo das ações é mínimo)

Busca em largura (BFS)

- O nó raiz é expandido primeiro, em seguida todos os sucessores do nó raiz são expandidos, depois os sucessores destes e assim sucessivamente
  - A BFS expande totalmente cada nível da árvore antes de partir para o próximo nível de profundidade
- É completa (se *b* e o espaço de estados são finitos)
- É ótima (se os custos de ação forem iguais)
- A complexidade assintótica de tempo é $O(b^d)$
- A complexidade assintótica de espaço é $O(b^d)$

  - *b é o fator de ramificação da árvore (quantidade de ações médias em cada estado)*
  - *d é a profundidade da solução mais rasa (solução mais próxima da raiz)*



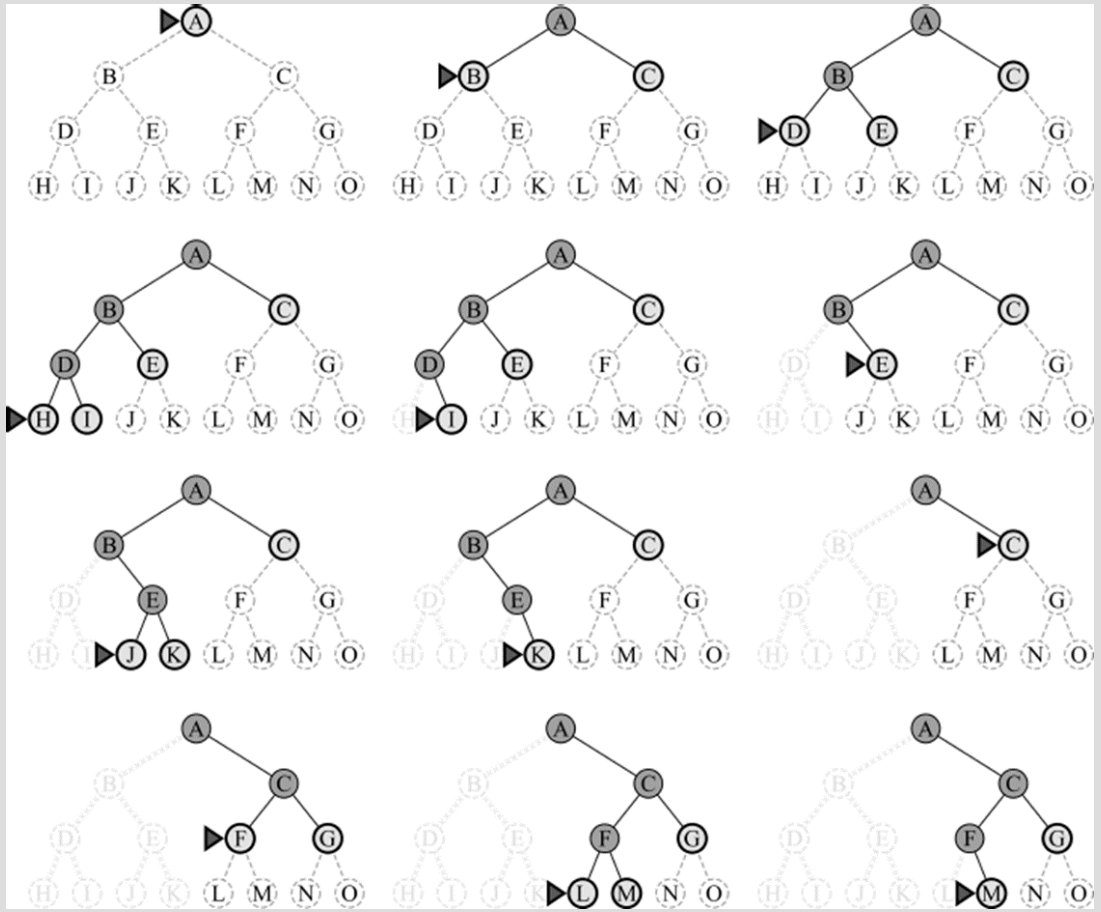
Busca de custo uniforme (UCS)

- A diferença para a busca em largura é que a UCS é aplicada em problemas cujas ações possuem custo diferente
  - Ao invés de expandir primeiro os nós mais próximos da raiz, a UCS expande primeiro os nós de menor custo agregado (custo do caminho até aquele nó ou *g(n)*)
- Também conhecida como algoritmo de Dijkstra (quando todos os custos são positivos) ou algoritmo de Bellman-Ford (considera também custos de ações negativas e resolve o problema desde que não exista ciclo de custo negativo no grafo de busca)
- É completa (se *b* e o espaço de estados são finitos e todos os custos são positivos ou se não existem ciclos negativos)
- É ótima
- A complexidade assintótica de tempo é $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$
- A complexidade assintótica de espaço é $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$

  - *C* é o custo da solução ótima*
  - *ϵ (épsilon) é o menor custo dentre as ações possíveis em todo o espaço de estados*

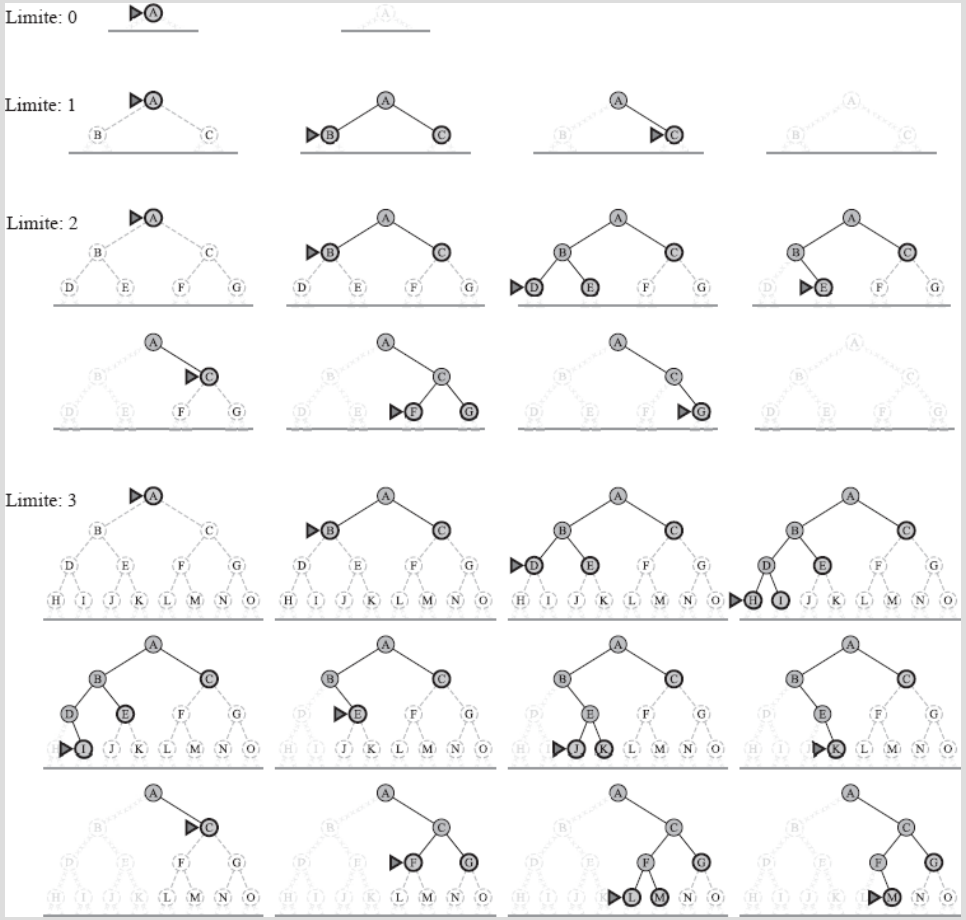
Busca em profundidade (DFS)

- Expande o nó mais profundo primeiro
- Não é completa (pode ficar presa em ciclos)
- Não é ótima
- A complexidade assintótica de tempo é $O(b^m)$
- A complexidade assintótica de espaço é $O(bm)$
  - *m é a profundidade máxima da árvore de busca*



Busca em aprofundamento iterativo (IDFS)

- Aplica DFS limitando a profundidade máxima da busca iterativamente
  - Começa com o limite = 0, realiza DFS, aumenta o limite em 1 e repete DFS sucessivamente até que o limite seja igual a *d*
- É completa (se *b* e o espaço de estados são finitos)
- É ótima (se os custos de ação forem iguais)
- A complexidade assintótica de tempo é $O(b^d)$
- A complexidade assintótica de espaço é $O(bd)$

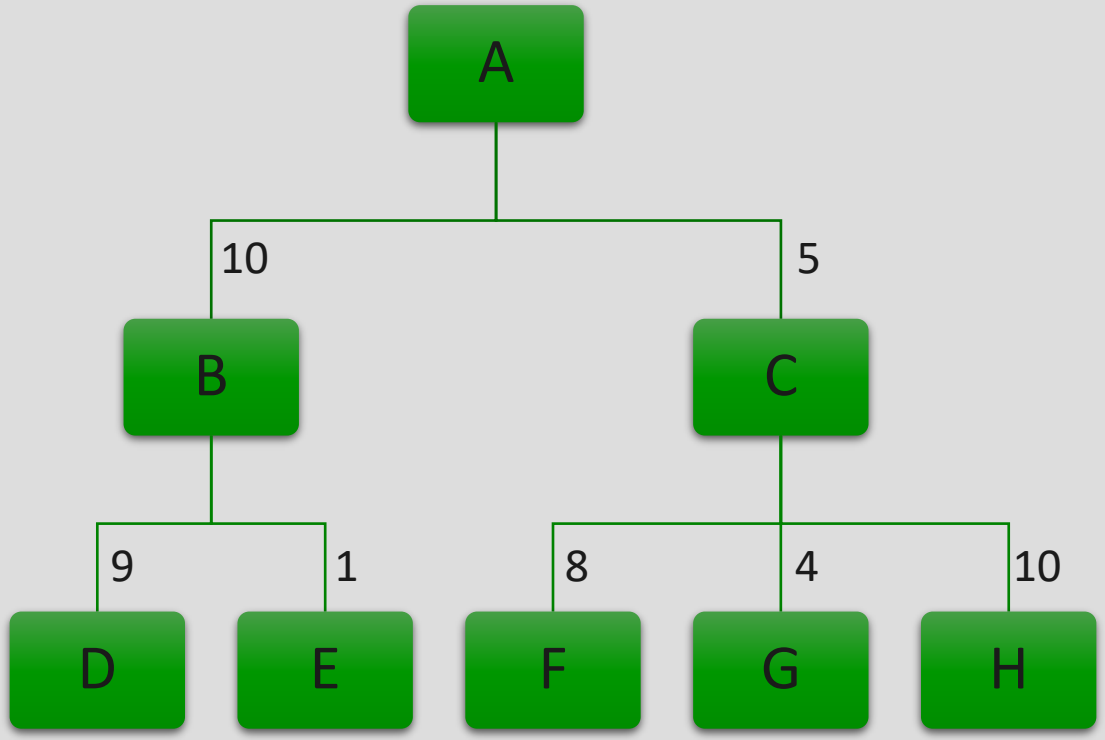


Busca bidirecional (BS)

- Aplica duas buscas, uma iniciando a partir da raiz e outra do estado meta (quando conhecido) simultaneamente
- A ideia de BS é que as buscas começam em extremos opostos e são finalizadas quando se encontram no “meio” do espaço de busca
- É completa (se usar a BFS ou a UCS)
- É ótima (se os custos de ação forem iguais e se usar a BFS ou a UCS)
- A complexidade assintótica de tempo é $O\left(b^{d/2}\right)$
- A complexidade assintótica de espaço é $O\left(b^{d/2}\right)$

Exercício 1 – Algoritmos de busca simples

- Encontre a sequência de nós expandida para a árvore de busca ao lado
  - Considere os custos dos caminhos apenas para a busca de custo uniforme
- BFS
- UCS
- DFS
- IDFS



Buscas informadas (heurísticas)

- Utiliza conhecimento do domínio do problema para direcionar a busca de forma mais eficiente no espaço de estados
- Aplica um função *h(n)* sobre os estados disponíveis para estimar o custo deste estado até o estado meta
  - Chamamos a função *h(n)* de função heurística
  - A função heurística é considerada admissível quando ela nunca superestima o custo necessário do estado avaliado até o estado meta, ou seja, uma heurística admissível sempre é otimista
    - Funções heurísticas admissíveis são fundamentais para garantir a otimalidade de vários algoritmos de busca, no entanto, as vezes é melhor usar heurísticas inadmissíveis quando o espaço de busca for muito grande (perdendo a garantia da otimalidade) mas garantindo maior eficiência computacional para os algoritmos de busca

Busca gulosa (BG)

- Expande sempre o nó cujo *h(n)* for menor
- É completa (se o espaço de estados é finito)
- A otimalidade da busca depende do problema e da heurística utilizada
- A complexidade assintótica de tempo e espaço dependem do problema e da heurística utilizada
  - Boas funções heurísticas em determinados problemas são $O(bm)$ tanto em tempo como espaço

Busca A* (A estrela)

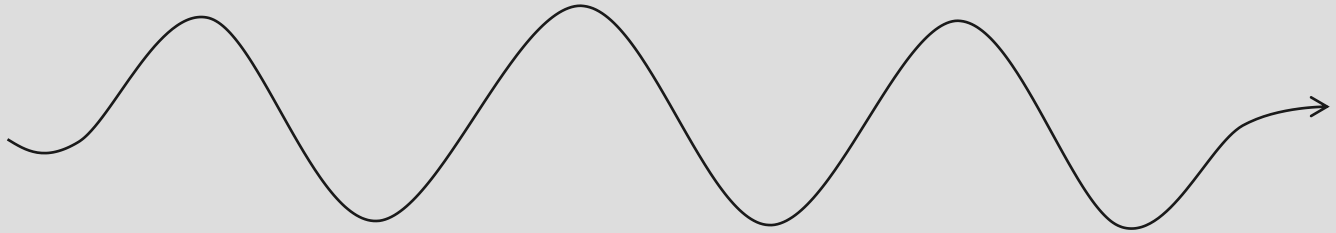
- Expande sempre o nó cujo *h(n) + g(n)* for menor
  - *g(n)* é o custo acumulado das ações até aquele estado (usado na BCU)
- É completa (se o espaço de estados é finito)
- É ótima (se *h(n)* for admissível)
- A complexidade assintótica de tempo é $O(b^d)$
- A complexidade assintótica de espaço é $O(b^d)$
- Embora a complexidade assintótica de tempo e espaço seja igual a da BL, na prática, a busca A* é muito mais eficiente se a heurística utilizada for boa

Exemplo da busca A* no jogo de 8 peças

Estado Inicial

1		3
4	2	6
7	5	8

g(n) = 0, h(n) = 3



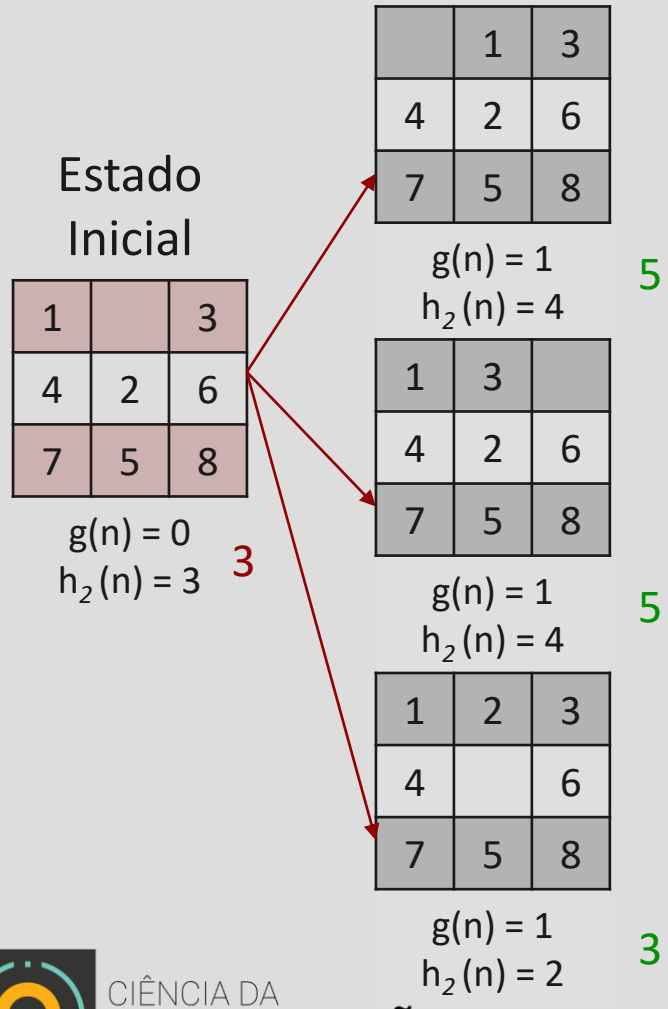
Estado Meta

1	2	3
4	5	6
7	8	

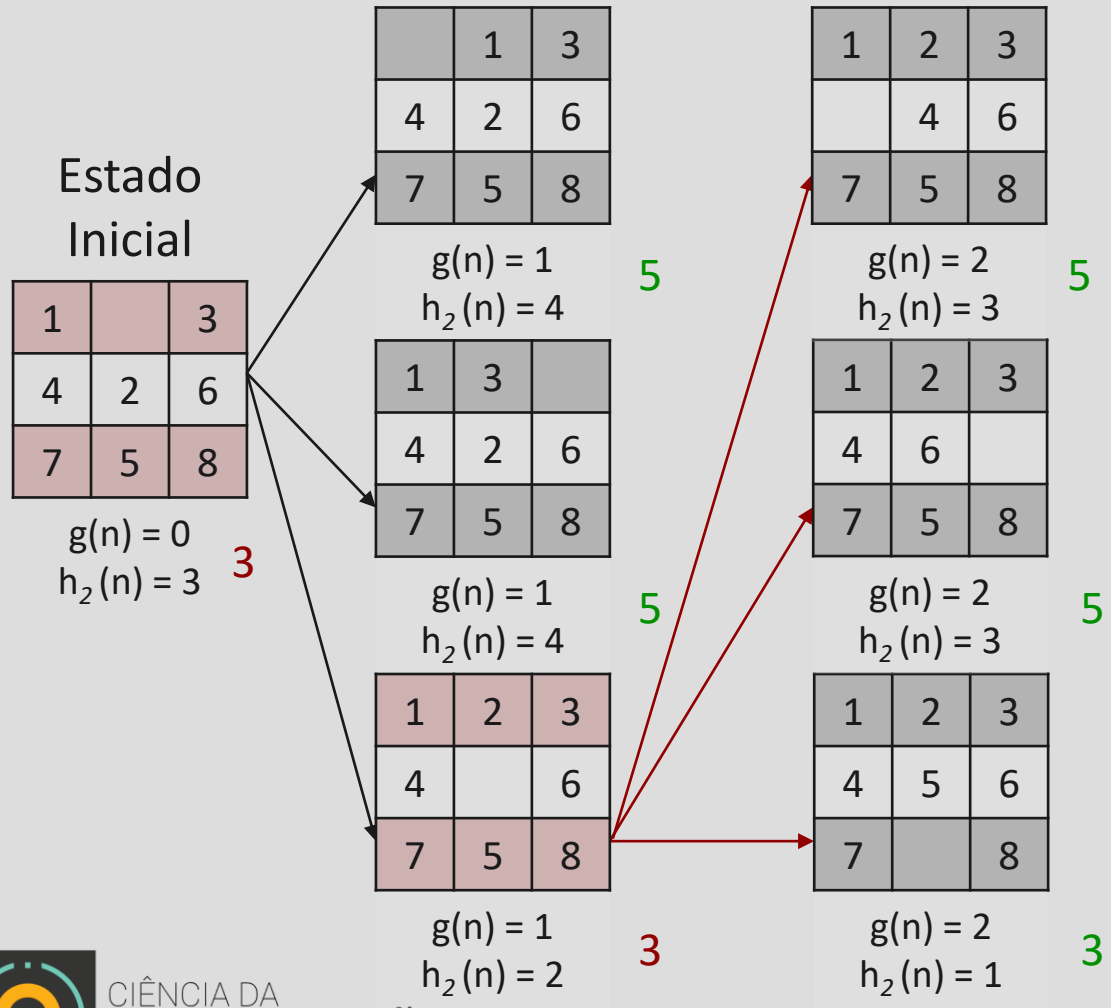
g(n) = ?, h(n) = 0

- $h_1(n)$ é a quantidade de peças fora do lugar
- $h_2(n)$ é a distância de Manhattan (distância de quarteirão)

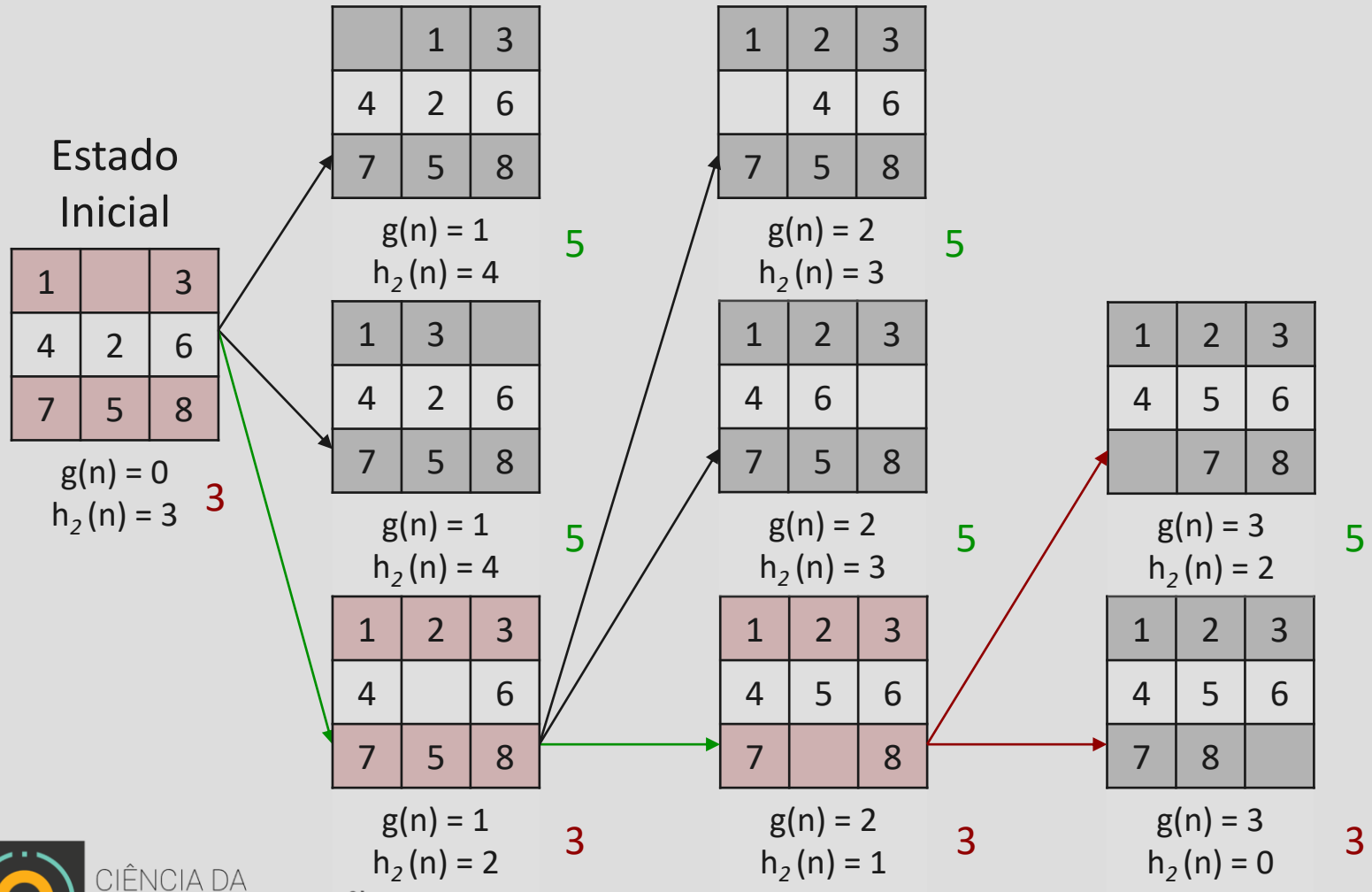
Exemplo da busca A* no jogo de 8 peças



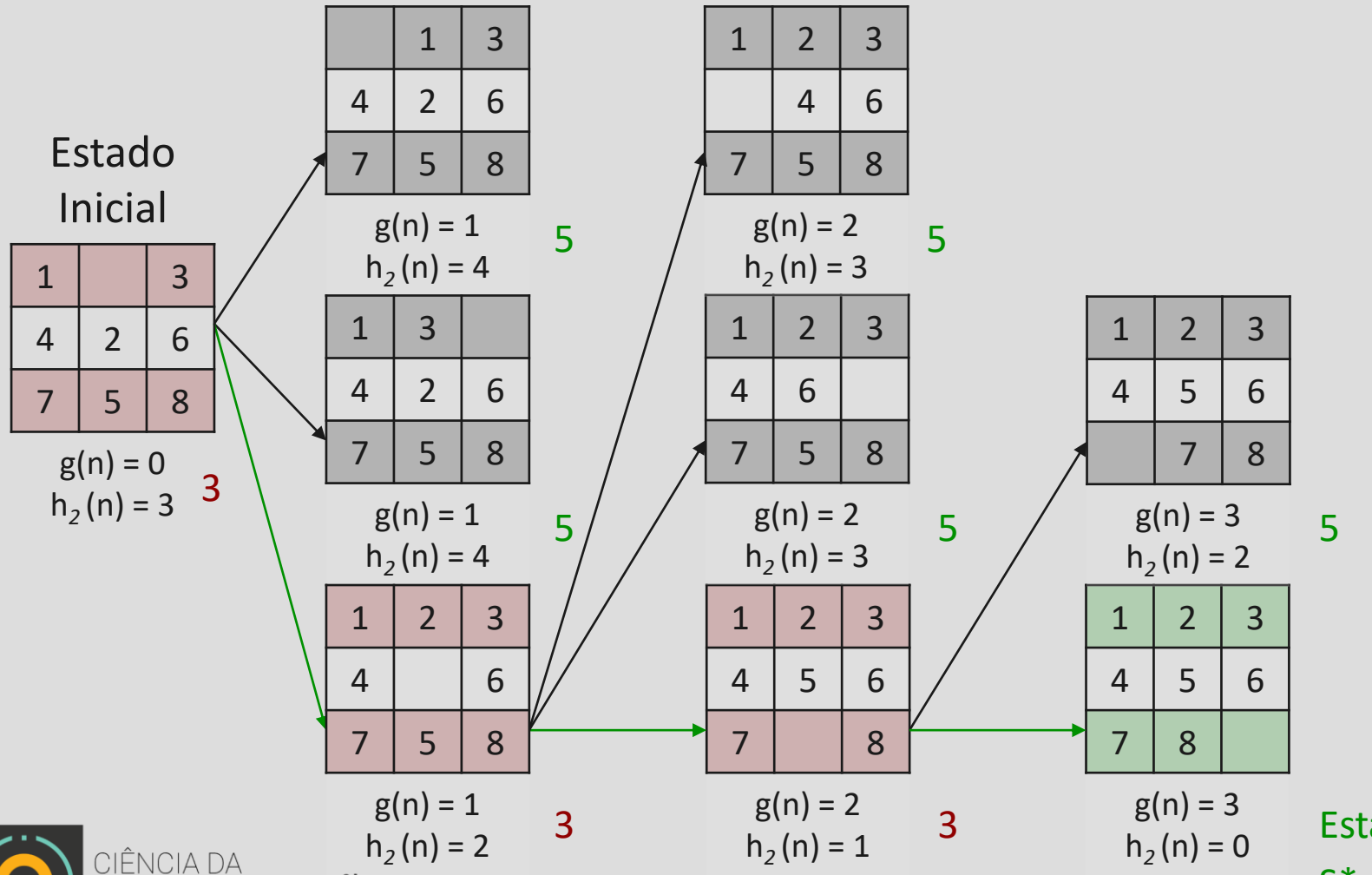
Exemplo da busca A* no jogo de 8 peças



Exemplo da busca A* no jogo de 8 peças



Exemplo da busca A* no jogo de 8 peças

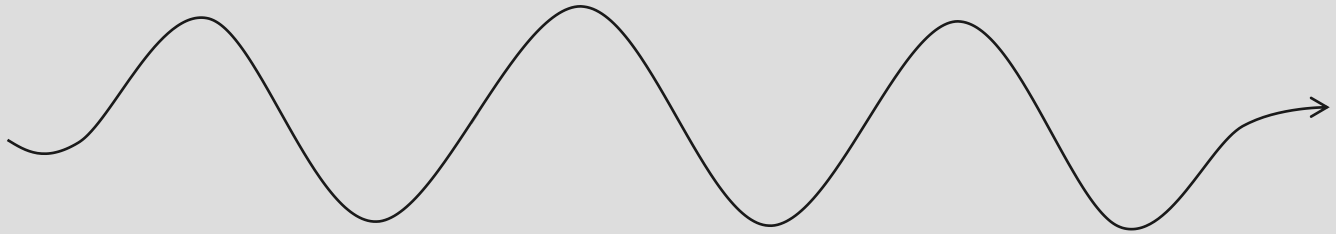


Atividade 1 – Busca A*

Estado Inicial

1	5	2
8		3
4	7	6

g(n) = 0, h(n) = 8



Estado Meta

1	2	3
4	5	6
7	8	

g(n) = ?, h(n) = 0

- Resolver o jogo de 8 peças com a busca A*
  - Usar a distância de distância de Manhattan como heurística (h(n))
  - Apresentar a árvore de busca gerada pelo algoritmo e o vetor com as ações para a solução ótima

Qual a melhor heurística?
Jogo de 8 peças

	Custo da busca (nós gerados)			Fator de ramificação efetivo		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2,01	1,42	1,34
8	368	48	31	1,91	1,40	1,30
10	1033	116	48	1,85	1,43	1,27
12	2672	279	84	1,80	1,45	1,28
14	6783	678	174	1,77	1,47	1,31
16	17270	1683	364	1,74	1,48	1,32
18	41558	4102	751	1,72	1,49	1,34
20	91493	9905	1318	1,69	1,50	1,34
22	175921	22955	2548	1,66	1,50	1,34
24	290082	53039	5733	1,62	1,50	1,36
26	395355	110372	10080	1,58	1,50	1,35
28	463234	202565	22055	1,53	1,49	1,36

Como criar uma heurística boa?

- Heurísticas admissíveis podem ser criadas para um problema ao considerarmos versões do problema com restrições relaxadas
  - Essa técnica pode ser usada inclusive para a criação automatizada de heurísticas
  - Quanto mais precisa a estimativa da heurística, mais eficiente torna-se o algoritmo de busca
- Considerando um conjunto de heurísticas admissíveis para um problema, aquela que tiver a estimativa de maior valor para um estado sempre será mais precisa e melhor que as demais
  - Podemos usar múltiplas heurísticas em conjunto em um mesmo problema, optando pela de maior valor em cada estado analisado

Outras estratégias

- Heurísticas inadmissíveis podem ser úteis quando o problema é muito complexo, quando as instâncias são muito grandes, ou quando computar uma heurística admissível é muito custos computacionalmente
  - Perde-se a garantia de otimalidade mas, em muitos problemas, existem heurísticas que tornam a busca muito eficiente em termos computacionais e trazem soluções de boa qualidade (muito próximas da qualidade da solução ótima)
- Uso de pontos de referência
  - Podemos fazer uso de estimativas precisas para um estado quando temos um banco de soluções de subproblemas comuns (banco de dados de padrões conhecidos)
- Uso de aprendizado
  - Podemos usar técnicas de aprendizado para definir pesos para as heurísticas
  - O método deixa de garantir a otimalidade mas pode trazer grandes avanços em eficiência para instâncias grandes sem prejudicar tanto a qualidade das soluções encontradas

Algoritmos de otimização

Agentes inteligentes que buscam encontrar uma solução “boa” de forma eficiente para problemas mais complexos

(RUSSELL; NORVIG, 2022. cap. 4; TALBI, 2009)

Algoritmos de otimização

- Os algoritmos de otimização são um subcampo da área de pesquisa operacional, teoria de otimização e teoria de controle ótimo
- A sua principal diferença para os algoritmos de busca simples e informadas são os problemas em que costumam ser aplicados
- Esses problemas de busca e otimização costumam ter as seguintes características:
  - Avaliar a qualidade e encontrar uma solução qualquer é trivial
  - Encontrar a solução ótima de forma exata leva tempo e/ou espaço de ordem exponencial ou pior pois o espaço de busca costuma ser gigantesco
    - A Programação Linear engloba alguns algoritmos que podem encontrar soluções ótimas, eficientemente na prática, para instâncias de problemas que tenham muitas restrições
  - Existem muitas soluções com custo próximo da ótima que são muito mais fáceis de se encontrar eficientemente
  - Existe uma versão do problema de otimização como problema de decisão e estes problemas são da classe NP-Difícil

Buscas Heurísticas

- Buscas heurísticas é um termo geralmente utilizado para definir as buscas que focam em eficiência computacional ao custo de não garantirem mais a solução ótima para todas as instâncias de um problema
- Sobretudo são buscas informadas que usam o contexto geral do problema para otimizar/guiar as escolhas da busca (estratégia ou heurística da busca) e, assim, reduzir consideravelmente o espaço de busca a ser analisado
- Podemos dividir as técnicas de busca heurística em três grupos principais, cada qual com um conjunto de técnicas de busca gerais chamadas de metaheurísticas:
  - Buscas Construtivas
  - Buscas Locais
  - Buscas por Recombinação

Buscas Construtivas

- Partem de uma estrutura de solução vazia
- Cria um conjunto de soluções parciais ou conjunto de componentes disponíveis para montar uma solução
- A cada passo (iteração), a busca aplica uma heurística na escolha de um componente desse conjunto que ainda não foi selecionado e o adiciona na solução final
- Exemplos de metaheurísticas construtivas:
  - Guloso, Guloso-k, Guloso-α
  - Construção repetida independente, Colônia de Formigas

Buscas Locais

- Partem de uma solução completa viável
  - Geralmente essa solução é construída com uma técnica de busca construtiva simples ou de forma aleatória
- Definem uma vizinhança
  - A vizinhança é um critério de similaridade/distância entre soluções
  - Exemplos de vizinhança:
    - 2-OPT, 3-OPT, Swap (troca), Shift (deslocamento)
- A cada passo, a busca navega dentro da vizinhança da solução atual (solução pivô) em busca de um nova solução pivô
- Exemplos de metaheurísticas locais:
  - Primeira melhora, Melhor melhora, Descida do gradiente (Subida de encosta)
  - Tabu, Têmpera Simulada

Busca Local – Exemplo: Têmpera Simulada

- Cria ou recebe uma solução inicial completa (S_0)
- Inicializa a melhor solução conhecida e a solução pivô ($S_{melhor} = S_0$ *e* $S_{pivo} = S_0$)
- Inicializa a temperatura ($t_0 = 1$)
- Inicia iteração da busca ($i = 0$)
  - Verifica critérios de parada para encerrar a busca
    - Tempo limite excedido
    - Solução pivô não sofreu alteração na última iteração
    - A melhor solução tem custo ótimo ($c(S_{melhor}) = c(S^*)$)
  - Enquanto houver soluções vizinhas da solução pivô atual ($\text{vizinho}(S_{pivo})$)
    - Computa variação de custo $\Delta c = \frac{c(S_{vizinha})}{c(S_{pivo})} - 1$
    - Muda solução pivô atual pela solução vizinha ($S_{pivo} = S_{vizinha}$) com probabilidade $P = \begin{cases} 1 & se \Delta c \leq 0 \\ e^{-\Delta c/t} & se \Delta c > 0 \end{cases}$
    - Se solução pivô mudou
      - Atualiza temperatura seguindo configurações de resfriamento ($t = resfriamento(t)$)
      - Verifica se nova solução pivô é melhor que a melhor solução conhecida e a atualiza se for o caso ($S_{melhor} = S_{pivo}$)
      - Pula para a próxima iteração de busca ($i = i + 1$)
- Retorna S_{melhor} e tempo de execução

Buscas por Recombinação

- Partem de um conjunto de soluções completas viáveis
  - Geralmente essas soluções são construídas de forma aleatória pois a busca depende de diversidade e representatividade de soluções
- Definem um operador de seleção
  - Esse operador é responsável por criar subconjuntos (normalmente pares) das soluções disponíveis levando em consideração algum critério
- Definem um operador de recombinação
  - Esse operador é responsável por combinar partes das soluções de um conjunto para formar um novo conjunto de soluções novas
- A cada passo, a busca aplica o operador de seleção e depois o de recombinação em cada conjunto selecionado, criando um novo conjunto de soluções completas
- Exemplos de metaheurísticas por recombinação:
  - Religamento de caminhos
  - Busca dispersa, feixe de partículas
  - Algoritmos genéticos e meméticos

Busca em ambientes complexos

Ambientes parcialmente observáveis, não determinísticos, sequenciais, dinâmicos, contínuos e desconhecidos

(RUSSELL; NORVIG, 2022. cap. 4)

Busca em ambientes parcialmente observáveis

- Quando a busca não consegue obter a informação completa do seu estado atual ela deve atuar com um conjunto de estados de crença atual
- A busca terá um passo adicional de previsão onde, baseado no seu conhecimento do modelo do ambiente, deverá determinar um conjunto de estados de crença possíveis para o estado atual real em que se encontra e planejar um conjunto de ações no qual maximize suas chances de sucesso
- A cada nova ação realizada o agente atualizará seu conjunto de estados de crença levando em conta os novos dados obtidos através do sensoriamento disponível no novo estado
- Os algoritmos de busca são completos (tem garantia de sucesso) em problemas onde todas as ações são reversíveis e o espaço de estados é finito, a um custo adicional de que algumas ações serão gastas na exploração dos estados (redução do conjunto de estados de crença)

Busca em ambientes não determinísticos

- Quando a busca não consegue ter certeza do resultado de sua ação ela deve atuar não mais com um conjunto pré-definido de ações fixas mas sim com um conjunto condicional de ações baseado em estados de crença futura
- A busca terá um passo adicional de previsão onde, baseado no seu conhecimento do modelo do ambiente, deverá determinar um conjunto de estados de crença possíveis para o próximo estado de cada ação planejada e suas chances de sucesso ou falha
- A busca deverá então prever um conjunto de ações condicionais para cada estado resultante previsto (estados de crença futura)
- Os algoritmos de busca são completos (tem garantia de sucesso) em problemas onde todas as ações são reversíveis e o espaço de estados é finito, a um custo adicional de que algumas ações serão gastas na correção de falhas

Busca em ambientes sequenciais e dinâmicos

- As buscas nesses ambientes devem armazenar e consultar os resultados obtidos no passado
- O modelo do ambiente deve constantemente ser atualizado com percepções passadas e atuais e as decisões devem se basear tanto nas percepções passadas como correntes
- Buscas nesses ambientes geralmente fazem uso de aprendizado por reforço para melhorar seu desempenho ao longo do tempo

Busca em ambientes contínuos

- Os problemas são representados por funções contínuas e muitas técnicas utilizadas são baseadas no cálculo numérico
- Em especial destacam-se:
  - técnicas de aproximação do gradiente de funções (gradiente empírico) que usam a descida de gradiente para encontrar mínimos/máximos de funções contínuas em espaços complexos
  - Programação linear (exemplo: algoritmo Simplex) e otimização com restrições
- Em alguns problemas é mais fácil o processo de discretização das variáveis contínuas
  - Nesse processo as variáveis contínuas são transformadas em categorias discretas para que seja possível utilizar as técnicas de busca convencionais de ambiente discretos

Busca em ambientes desconhecidos

- Também conhecidas como técnicas de busca online pois os agentes inteligentes devem, através de interações do tipo ação-percepção do resultado, explorar ou mapear primeiro o seu ambiente para então planejar suas ações
- Os algoritmos de busca são completos (tem garantia de sucesso) em problemas que podem ser explorados com segurança, onde todas as ações são reversíveis e onde o espaço de estados é finito
- A eficiência da busca é avaliada com sendo sua razão competitiva, ou seja, se comparada com uma busca que tivesse o ambiente já conhecido qual seria o custo adicional necessário gasto na exploração para se obter o mesmo resultado

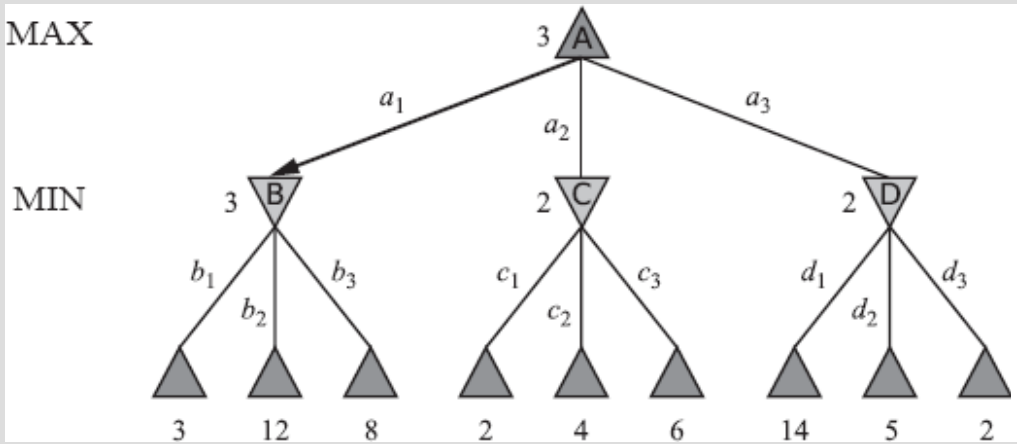
Busca competitiva e jogos

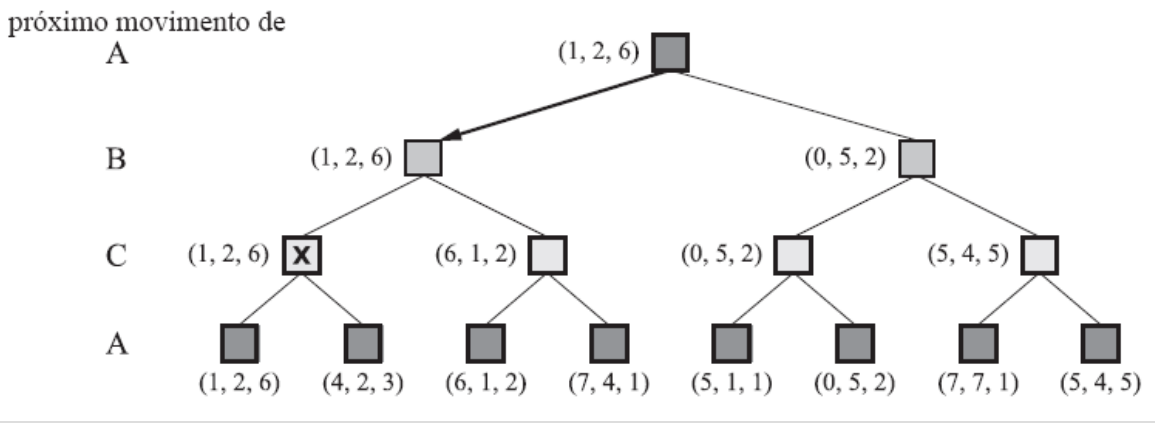
Ambientes multiagentes

(RUSSELL; NORVIG, 2022. cap. 5)

Busca MINMAX

- Algoritmo recursivo que percorre todo o caminho até as folhas da árvore (BP) e propaga os valores do resultado do jogo de volta pela árvore à medida que a recursão retorna
- Ele possui as mesmas características de complexidade assintótica de uma BP pois também necessita percorrer toda a árvore de busca, o que o torna impraticável para problemas mais complexos

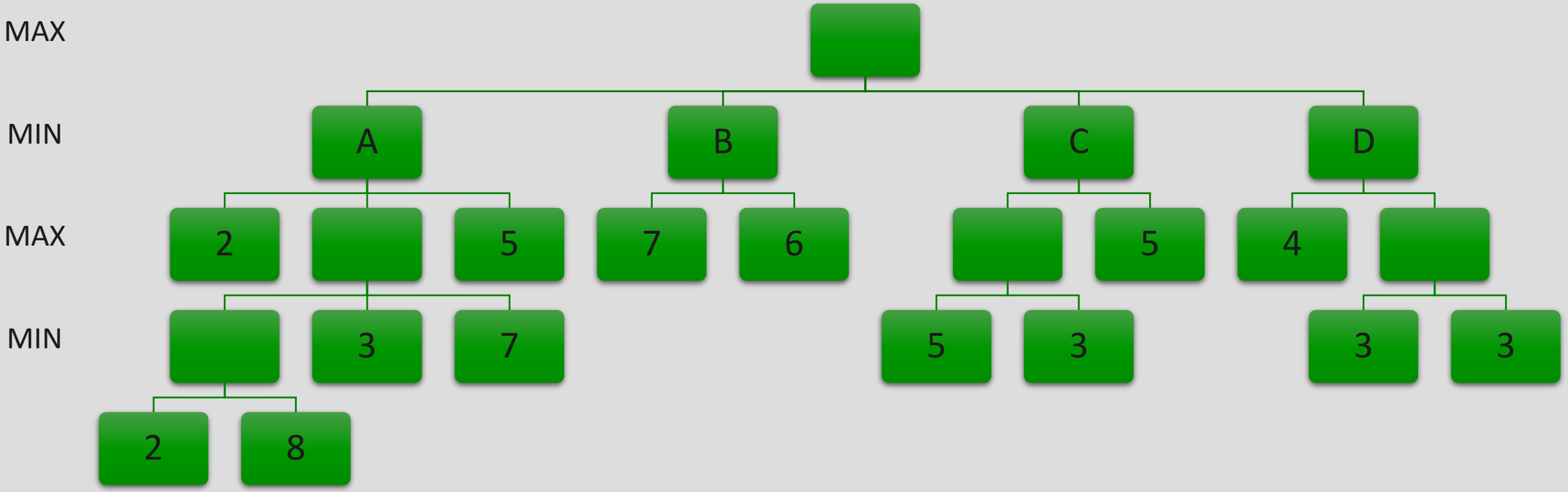




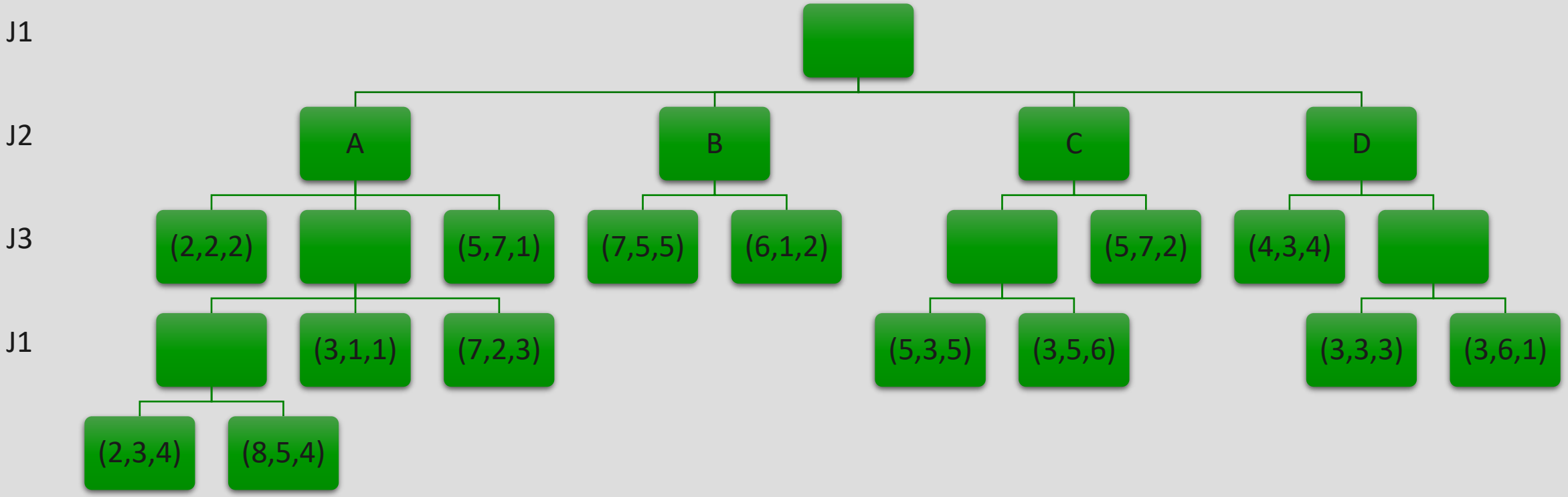
Exercício 2 e 3 – Busca MINMAX

- Encontre os valores MINMAX para cada nó da árvore e indique a próxima ação (A, B, C ou D) que será indicada pelo algoritmo para o jogador principal (MAX) ou o jogador atual (J1)
- As árvores dos exercícios se encontram nos slides seguintes

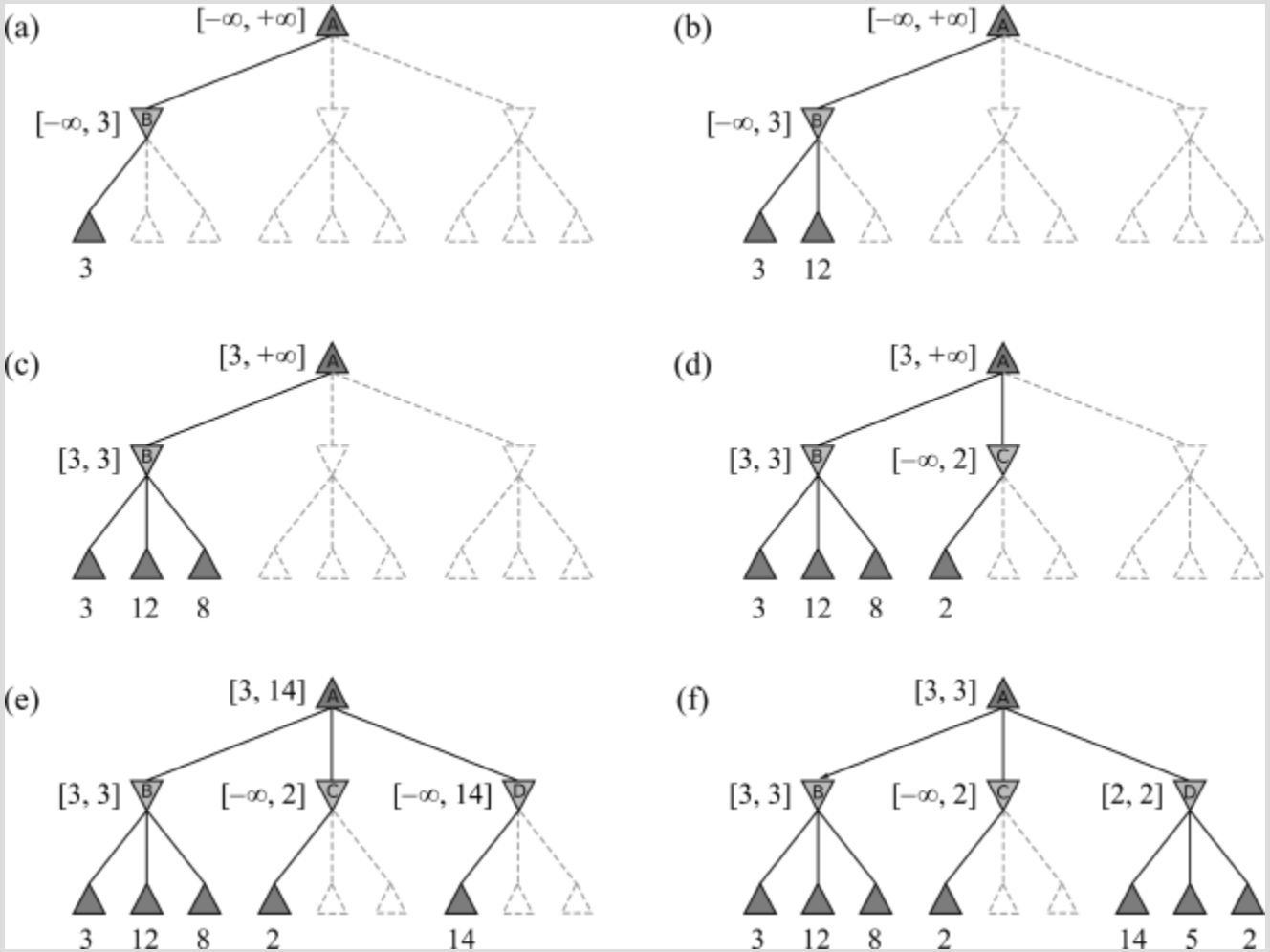
Exercício 2 – Busca MINMAX (2 jogadores)



Exercício 3 – Busca MINMAX (3 jogadores)



Busca MINMAX – Poda alfabeta



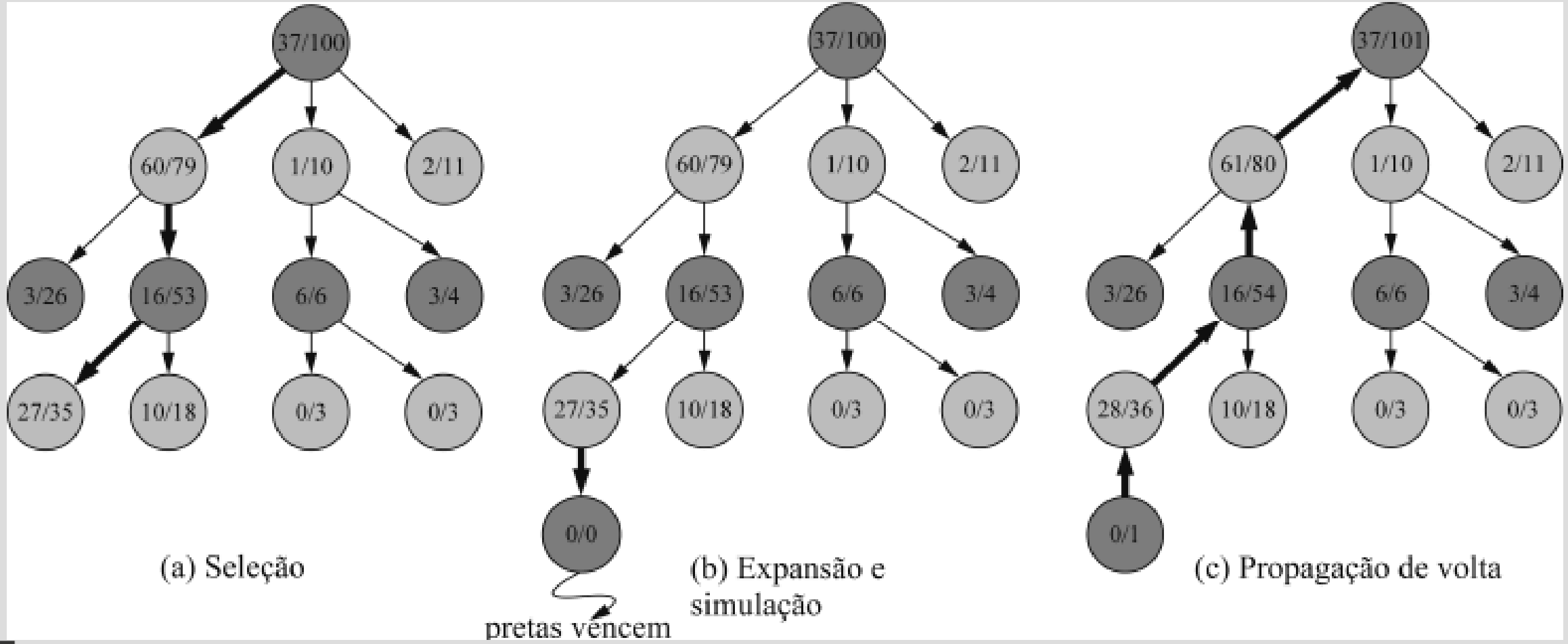
Busca MINMAX – otimizações

- Usando apenas a busca MINMAX com uma função de avaliação e testes de corte razoáveis em um jogo como o Xadrez, onde a ramificação média é cerca de 35, conseguimos executar uma busca em profundidade máxima de 5 com os computadores atuais dentro de um tempo realista em uma competição (desempenho similar ao de um jogador nível médio)
- Se usarmos a poda alfabeta com uma tabela de transposição para eliminar estados repetidos conseguimos chegar a uma profundidade 14 (nível equiparável ao de um humano especialista)
- Os melhores programas de Xadrez como o STOCKFISH alcançam a profundidade de 30 ou mais na árvore de busca, o que ultrapassa bastante a capacidade de qualquer jogador humano (em torno de 15 a 20 jogadas para um grande mestre). Para isso, fazem uso de uma função de avaliação extensivamente ajustada e de um banco de dados de fim de jogo para conjuntos de jogadas em estados finais
- Para um jogo de alta ramificação como o Go, onde a ramificação inicial começa em 361, ou para aqueles que se desconhece uma função de avaliação eficiente, a busca MINMAX pode ser totalmente ineficaz

Busca em árvore de Monte Carlo (MCTS)

- Recebe o nome devido ao Cassino de Monte Carlo em Mônaco pois os algoritmos de Monte Carlos são algoritmos randomizados
- O algoritmo BAMT realiza uma série de simulações do jogo, equilibrando entre explotação e exploração, do jogo e aprende com os resultados obtidos
  - Explotação – focar nos estados que funcionaram bem nas simulações anteriores
  - Exploração – focar nos estado que tiveram menos simulações até o momento
- É um algoritmo no estilo de várias técnicas do aprendizado por reforço pois constrói sua árvore de busca através da técnica de *self-play* (joga com ele mesmo)

Busca em árvore de Monte Carlo



Avaliação de modelos de busca

Inferência estatística: características de uma População a partir de Amostras

(BARBETTA; REIS; BORNIA, 2010. cap. 6-9)

Conceitos básicos: população e amostra

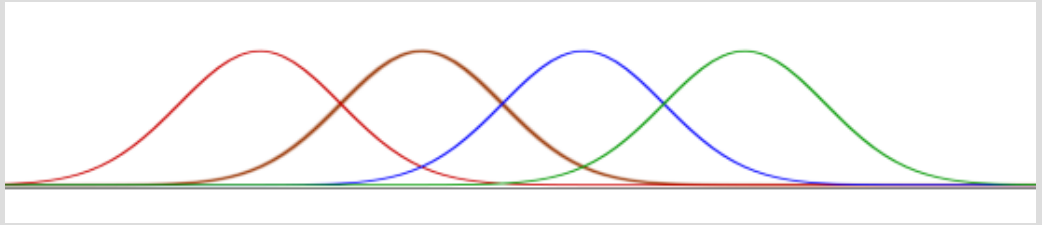
- População é o conjunto completo de elementos pesquisados
  - Totalidade de participantes com a característica pesquisada
  - As características conhecidas de uma população são conhecidas como parâmetros da população
- Amostra é um subconjunto da população
  - Parte menor da população cujo dados serão coletados pela pesquisa
  - As características mensuráveis obtidas a partir de uma amostra são conhecidas como estatísticas ou estimadores

Característica	Parâmetro (população)	Estatística (amostra)
Média	μ	$\bar{X}\ ou\ \bar{x}$
Desvio Padrão	σ	$S\ ou\ s$
Variância	σ^2	$S^2\ ou\ s^2$
Número de elementos	N	n

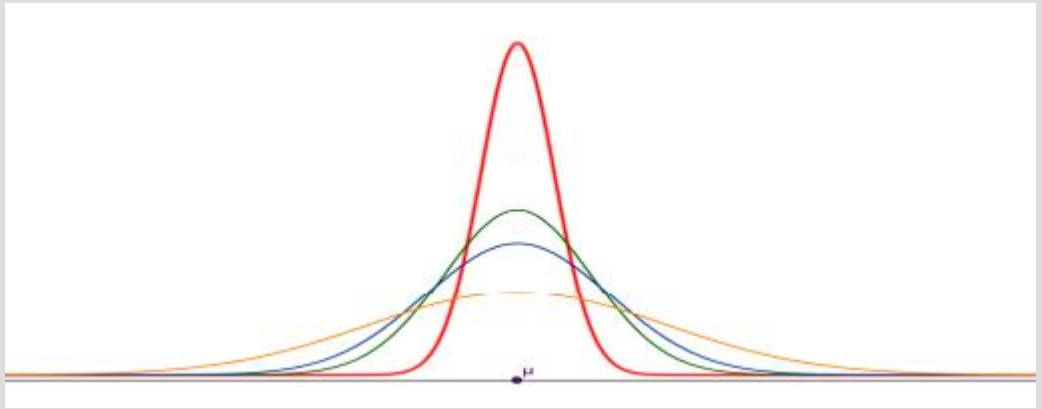
Conceitos básicos: distribuição Gaussiana

- Também conhecida como Normal, aproxima a função binomial, que é discreta, através de uma variável contínua (x)
  - A função binomial representa a quantidade de "sucessos" em n repetições independentes de um experimento (tamanho amostral)
- $f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
- Com $-\infty < x < \infty$ e $-\infty < \mu < \infty$ e $\sigma > 0$
- A média desloca a função no eixo x e determina o centro da função (que é simétrica)
- O desvio padrão espreme (quando menor) ou alonga a curva

Diferentes médias, desvio padrão igual



Desvio padrão diferente, médias iguais



Inferência estatística

- É o processo de inferir características de uma população por meio da observação de uma amostra
- Utilizada quando conhecemos as características da amostra mas não da população
- Chamado também de raciocínio indutivo (particular para geral ou generalização)
- Contrapõe o raciocínio dedutivo (geral para o particular), onde conhecemos as características da população e queremos estimar as características amostrais usando técnicas probabilísticas

Teorema do Limite Central

- Uma amostra aleatória simples de uma população com média μ e variância σ^2 converge para $\bar{x} = \mu$ e $s^2/n = \sigma^2$ quando $n \to \infty$
- Em geral, mostra-se que para amostras com n > 10 demostram uma aproximação considerada razoável
- Considerando o teorema é possível afirmar que quando o tamanho amostral é suficientemente grande, a distribuição da média amostral é uma distribuição aproximadamente normal
- Vídeo explicativo – But what is the Central Limit Theorem? (SANDERSON, 2023)

Intervalo de confiança

- Com probabilidade alta (em geral, indicada por 1-α), o intervalo [$\bar{x}$ - erro, $\bar{x}$ + erro] conterá o verdadeiro valor da média populacional μ
- Exemplo: se 1-α = 0,95 em 95% das amostras possíveis o intervalo obtido conterá o verdadeiro valor do parâmetro
- O intervalo pode ser computado através da fórmula: $\left[\bar{x} \pm z\frac{s}{\sqrt{n}}\right]$, onde z é a abscissa da curva normal padrão que deixa probabilidade (área) igual a α acima dela

Intervalo de Confiança	z
80%	1,28
85%	1,44
90%	1,64
95%	1,96
99%	2,57
99,5%	2,80
99,9%	3,29

Aprendizagem de máquina

Melhorando o desempenho da máquina e seus algoritmos através de observações autônomas do ambiente

(RUSSELL; NORVIG, 2022. cap. 19-22)

Sistemas especialistas

- Esses sistemas dependiam exclusivamente do conhecimento de especialistas humanos sobre o domínio do ambiente para desempenharem suas operações através de regras-condições pré-estabelecidas programaticamente
  - Ainda são largamente utilizados como alternativas simplificadas para a resolução de problemas computacionais diversos onde já são conhecidas boas e eficientes soluções algorítmicas
- Predominantes durante as décadas de 70 e 80, gradualmente foram sendo substituídos por técnicas de aprendizado de máquina, que exigiam máquinas com maiores recursos computacionais mas que possibilitavam a resolução de instâncias e problemas mais complexos sem intervenção humana direta ou necessidade de especialistas no domínio
  - Vale ressaltar que especialistas de um domínio, quando disponíveis, podem e continuam auxiliando as técnicas de aprendizado, tornando-as mais eficientes e menos custosas computacionalmente

Aprendizado

- Qualquer componente de um sistema ou algoritmo pode ser melhorado através do aprendizado de máquina sem a necessidade de conhecimento prévio por parte do agente inteligente
- O aprendizado de máquina pode ser dividido em três grandes grupos quanto as técnicas utilizadas e tipos de problemas resolvidos
  - Aprendizado supervisionado: o agente usa dados rotulados para criar um modelo que mapeia novas entradas em saídas. Usado em problemas de classificação e regressão.
  - Aprendizado não-supervisionado: o agente cria um modelo que mapeia relacionamentos entre os dados sem qualquer rótulo previamente estabelecido. Usado em problemas de agrupamento (*clustering*), redução de dimensionalidade e geração de novos dados.
  - Aprendizado por reforço: o agente cria um modelo do ambiente a partir de uma série de recompensas e punições recebidas (observadas) em ambientes simulados. Usado em problemas de controle onde o agente interage diretamente com o ambiente do problema

Aprendizado supervisionado

Observa exemplos de pares de entradas e saídas e cria um mapeamento entre elas

(RUSSELL; NORVIG, 2022. cap. 19-21)

Regressão linear

- Vantagens
  - Computacionalmente simples, robusto e fácil de interpretar
  - Fundamentado em conceitos de cálculo (funções e derivadas)
- Desvantagens
  - Seu uso é restrito a dados linearmente dependentes
- Treinamento
  - Usa um conjunto de dados de treinamento rotulado (X, y) para aprender os pesos (W) adequados através da técnica de gradiente descendente

Regressão linear

- O objetivo de uma regressão é aprender um modelo que representa um conjunto de entradas.
- Uma regressão é linear quando o modelo gerado pode ser representado através de uma função de primeiro grau, na forma

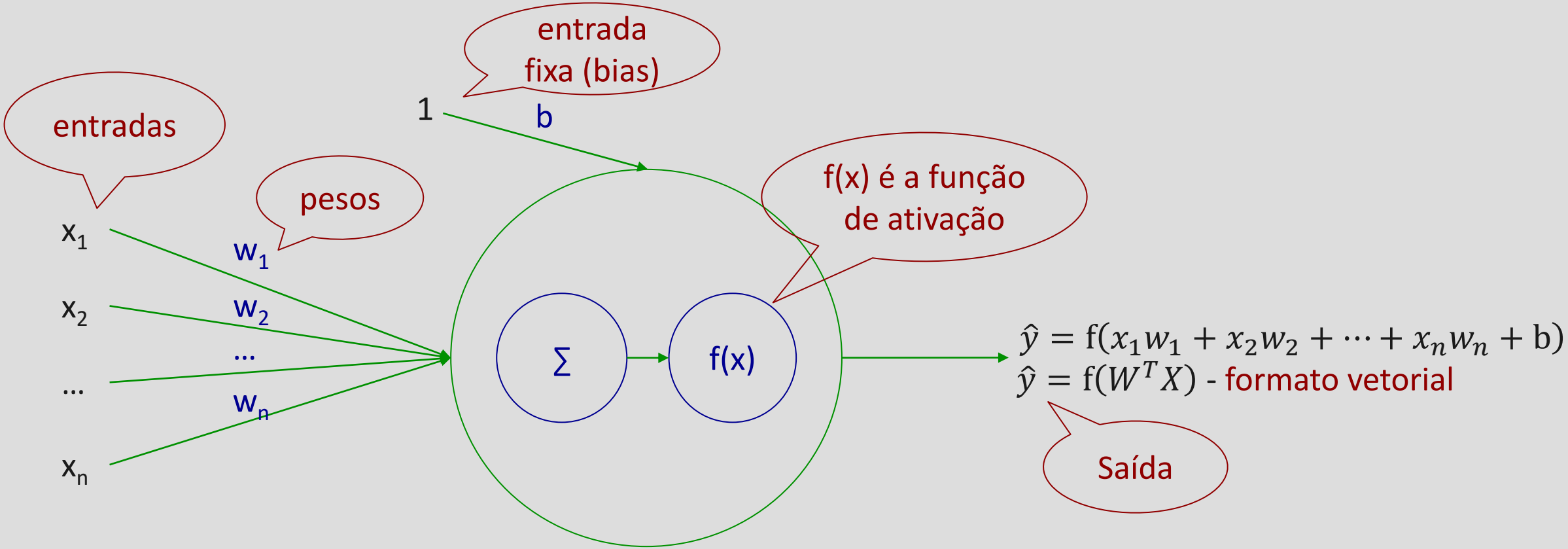
$$\hat{y} = W^T X + b$$

Variável	Descrição
$\hat{y}$	Predição, $\hat{y} \in \mathbb{R}^N$ Modelo de y
X	Entradas, $X \in \mathbb{R}^{NXd}$
d	Dimensão das entradas (quantidade de atributos)
W^T	Vetor de pesos transposto, $W \in \mathbb{R}^d$ (coeficiente angular)
b	Bias, $b \in \mathbb{R}$ (coeficiente linear ou termo independente)
N	Número de instâncias de entrada

Redes neurais artificiais (RNA)

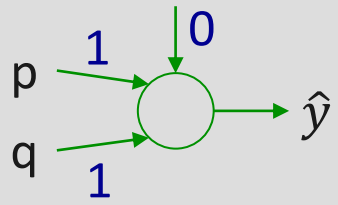
- Vantagens
  - Turing completo, aproximador universal de funções, flexível e gera modelos altamente eficientes em termos de custos computacionais
  - Fundamentado em conceitos de cálculo (funções e derivadas)
- Desvantagens
  - Dificuldade na estimativa dos meta-parâmetros (quantidade de neurônios, tipos, camadas, entre outros), treinamento mais custoso computacionalmente e modelos difíceis de serem interpretados (caixa preta)
- Treinamento
  - Usa um conjunto de dados de treinamento rotulado (X, y) para aprender os pesos (W) adequados através da técnica de gradiente descendente (backpropagation)

Neurônio artificial

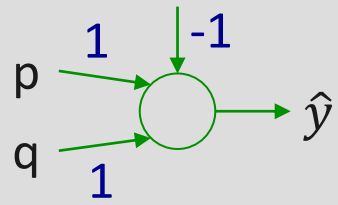


Exercício 4 – RNA

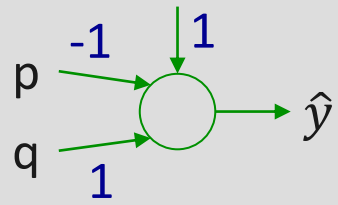
r1



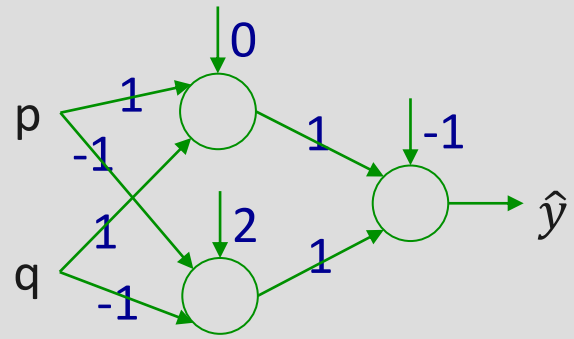
r2



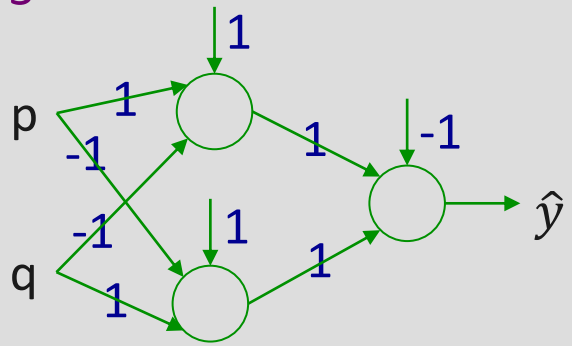
r3



r4



r5

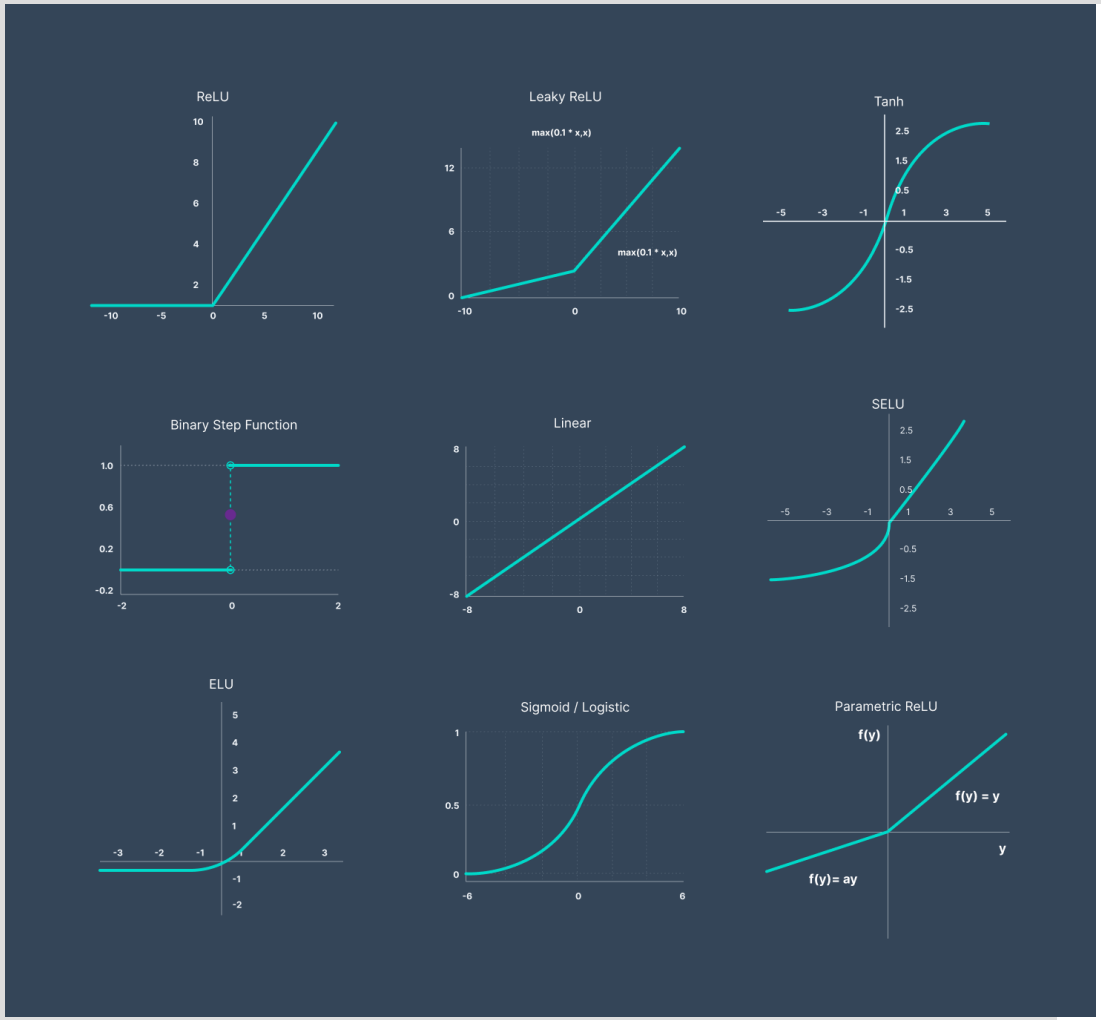


$$f = \begin{cases} 1 \; se \; x > 0 \\ 0 \; se \; x \leq 0 \end{cases}$$

p	q	r1	r2	r3	r4	r5
0	0					
0	1					
1	0					
1	1					

$$r1 = p \quad q$$
$$r2 = p \quad q$$
$$r3 = p \quad q$$
$$r4 = p \quad q$$
$$r5 = p \quad q$$

Funções de ativação



ReLU

$$f(x) = max\,(0, x)$$

Leaky ReLU

$$f(x) = max\,(0.1x, x)$$

Tanh

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

Binary step

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

Linear

$$f(x) = x$$

SELU

$$f(\alpha, x) = \lambda \begin{cases} \alpha(e^{x} - 1) & for\ x < 0 \\ x & for\ x \geqslant 0 \end{cases}$$

ELU

$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha(e^{x} - 1) & for\ x < 0 \end{cases}$$

Sigmoid / Logistic

$$f(x) = \frac{1}{1 + e^{-x}}$$

Parametric ReLU

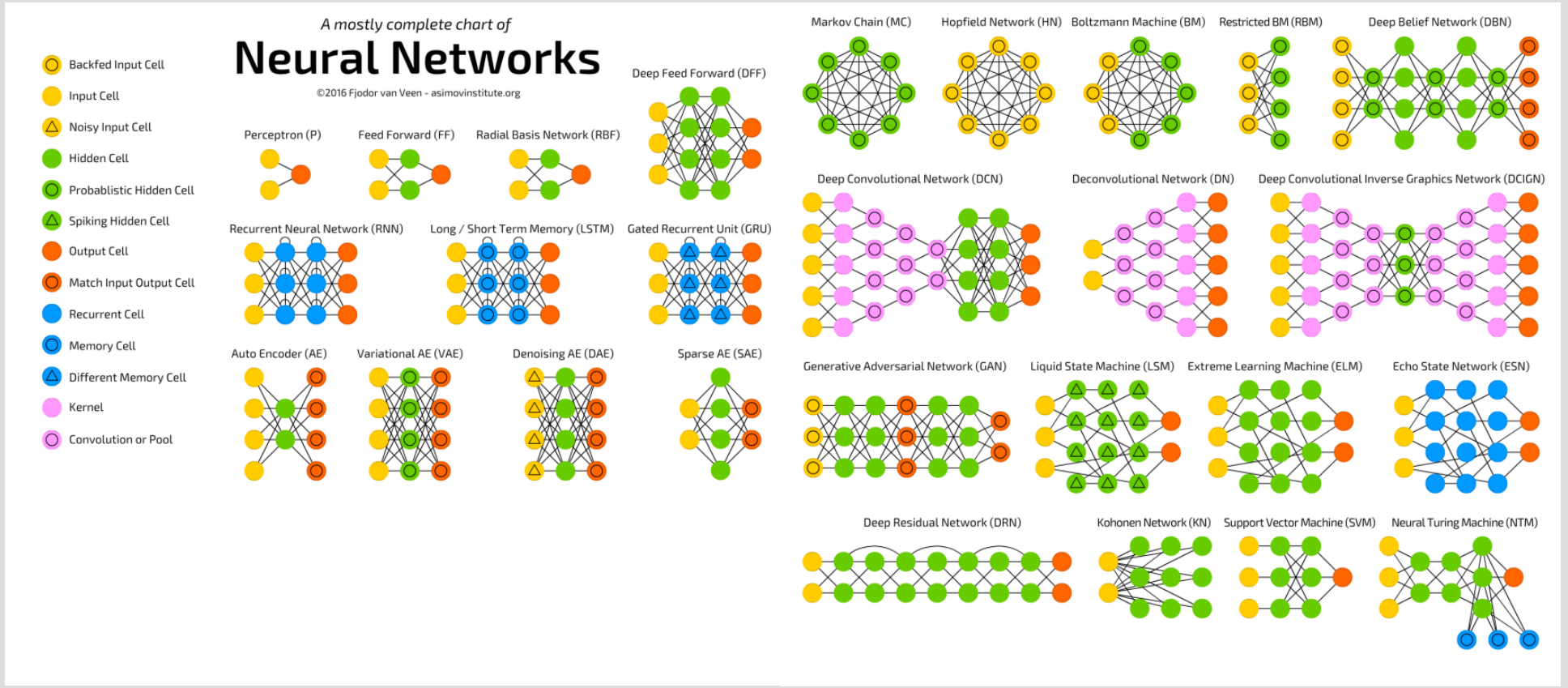
$$f(x) = max\,(ax, x)$$

Softmax

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

V7 Labs

Tipos de RNA


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Árvores de decisão (AD)

- Vantagens
  - Robusto e gera modelos facilmente interpretáveis (caixa branca)
  - Fundamentado em técnicas estatísticas e probabilísticas
- Desvantagens
  - Os modelos podem ficar extensos e serem pouco flexíveis (especialmente em problemas de regressão)
- Treinamento
  - Usa um conjunto de dados de treinamento rotulado (X, y) para aprender regras usando a entropia para computar o ganho de informação (escolhe o atributo que representa maior ganho de informação ou que gera os conjuntos mais puros a cada divisão da árvore)

Árvores de decisão

- A entropia mede a incerteza de uma variável aleatória e pode ser usada para definir o grau de incerteza de um conjunto

$$Entropia(S) = -\sum_i p_i \log_2 p_i$$

- O ganho de informação é a redução na entropia dado um atributo

$$Ganho(S, A)$$
$$= Entropia(S) - \sum_a \frac{N_a}{N} Entropia(S_a)$$

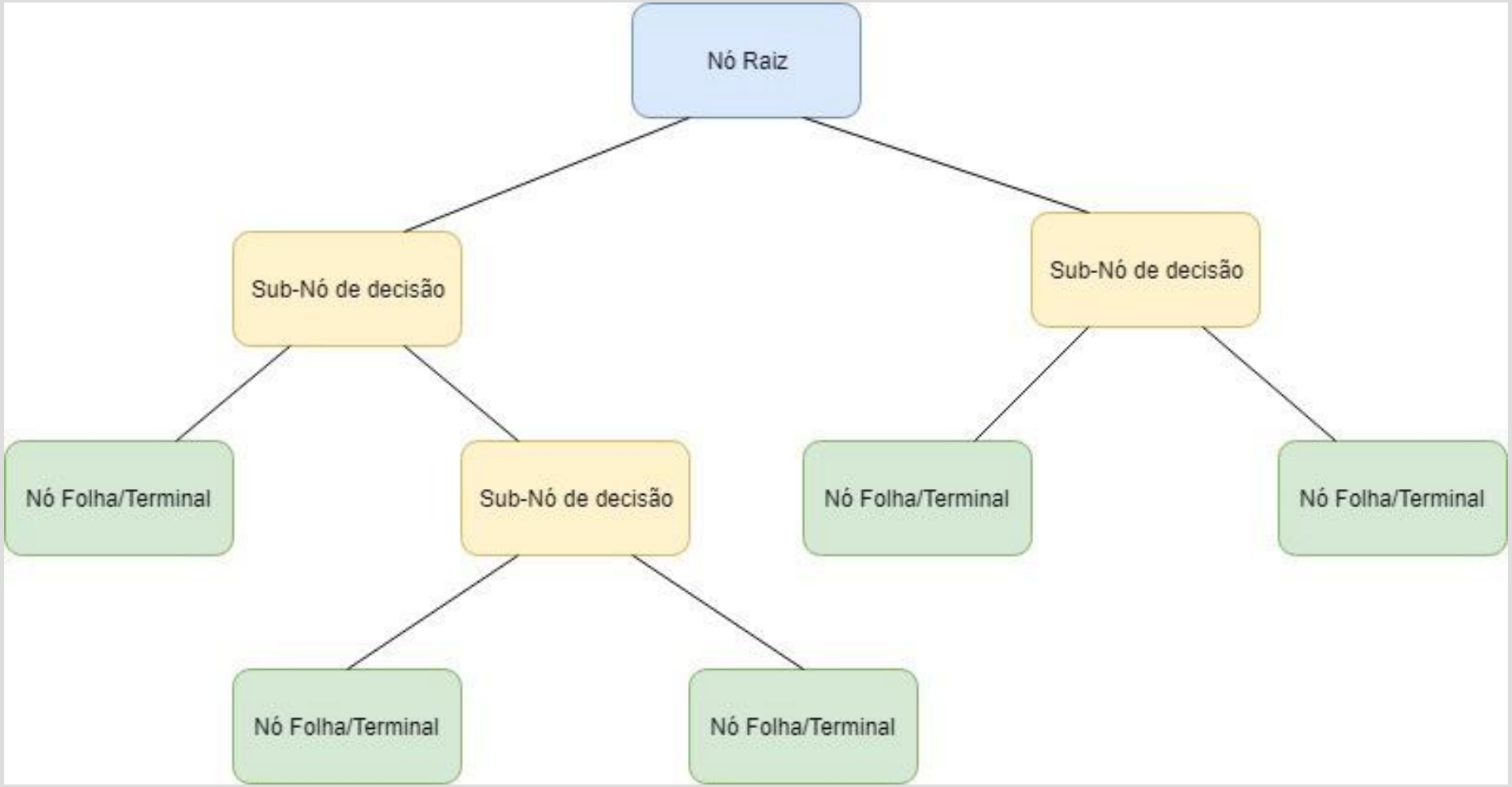
- A pureza de um conjunto pode ser medida com índice de Gini

$$Gini(S_a) = 1 - \sum_i {p_i}^2$$

$$Gini(S, A) = \sum_a \frac{N_a}{N} Gini(S_a)$$

Variável	Descrição
S	Conjunto de variáveis
S_a	Conjunto de variáveis com o atributo a
i	Identificador da classe (grupo)
p_i	Probabilidade de uma amostra do conjunto S ser da classe i
A	Atributo de uma classe
a	Identifica um valor (manifestação de um atributo A
N	Número de instâncias do conjunto S
N_a	Número de instâncias do conjunto S com o valor a

Protótipo de uma árvore de decisão



AD – Técnica da entropia e ganho

- Computa-se o ganho de cada atributo e escolhe-se o atributo com o maior ganho
  - O processo se repete a cada divisão da árvore de decisão

$$Entropia(S) = -\sum_i p_i \log_2 p_i = -\left(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6}\right) = 0.92$$

$$Entropia(Alto) = -\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right) = 0.91$$

$$Entropia(Normal) = -\left(\frac{0}{1}\log_2\frac{0}{1} + \frac{1}{1}\log_2\frac{1}{1}\right) = 0$$

$$Entropia(Baixo) = -\left(\frac{0}{2}\log_2\frac{0}{2} + \frac{2}{2}\log_2\frac{2}{2}\right) = 0$$

$$Ganho(S, A1) = Entropia(S) - \sum_a \frac{N_a}{N} Entropia(S_a) = 0.92 - \left(\frac{3}{6}\times 0.91 + \frac{1}{6}\times 0 + \frac{2}{6}\times 0\right) = 0.46$$

A1	A2	Classe
Alto	Muito	1
Baixo	Pouco	2
Baixo	Muito	2
Normal	Muito	2
Alto	Pouco	1
Alto	Muito	2

AD – Técnica do índice de Gini

- Computa-se o índice Gini de cada atributo e escolhe-se o atributo com o menor índice (maior pureza)
  - O processo se repete a cada divisão da árvore de decisão

$$Gini(Alto) = 1 - \sum_i {p_i}^2 = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = 0{,}44$$

$$Gini(Normal) = 1 - \left(\frac{0}{1}\right)^2 - \left(\frac{1}{1}\right)^2 = 0$$

$$Gini(Baixo) = 1 - \left(\frac{0}{2}\right)^2 - \left(\frac{2}{2}\right)^2 = 0$$

$$Gini(S, A1) = \sum_a \frac{N_a}{N} Gini(S_a) = \frac{3}{6} \times 0{,}44 + \frac{1}{6} \times 0 + \frac{2}{6} \times 0 = 0{,}22$$

A1	A2	Classe
Alto	Muito	1
Baixo	Pouco	2
Baixo	Muito	2
Normal	Muito	2
Alto	Pouco	1
Alto	Muito	2

Exercício 5 – AD

- Construa 2 árvores de decisão, uma usando o método de seleção de entropia/ganho e a outra usando o índice de Gini, para o conjunto de entradas abaixo

Clima	Temperatura	Humidade	Vento	Futebol
Chuvoso	Frio	Alta	Intenso	Não
Ensolarado	Agradável	Baixa	Fraco	Sim
Nublado	Calor	Normal	Fraco	Sim
Nublado	Agradável	Normal	Fraco	Sim
Ensolarado	Calor	Normal	Fraco	Sim
Nublado	Frio	Normal	Intenso	Não
Chuvoso	Calor	Alta	Intenso	Sim
Chuvoso	Agradável	Alta	Intenso	Não
Ensolarado	Agradável	Baixa	Intenso	Sim
Nublado	Frio	Baixa	Fraco	Não

Aprendizado não supervisionado

Observa exemplos de entradas e cria relacionamentos entre eles

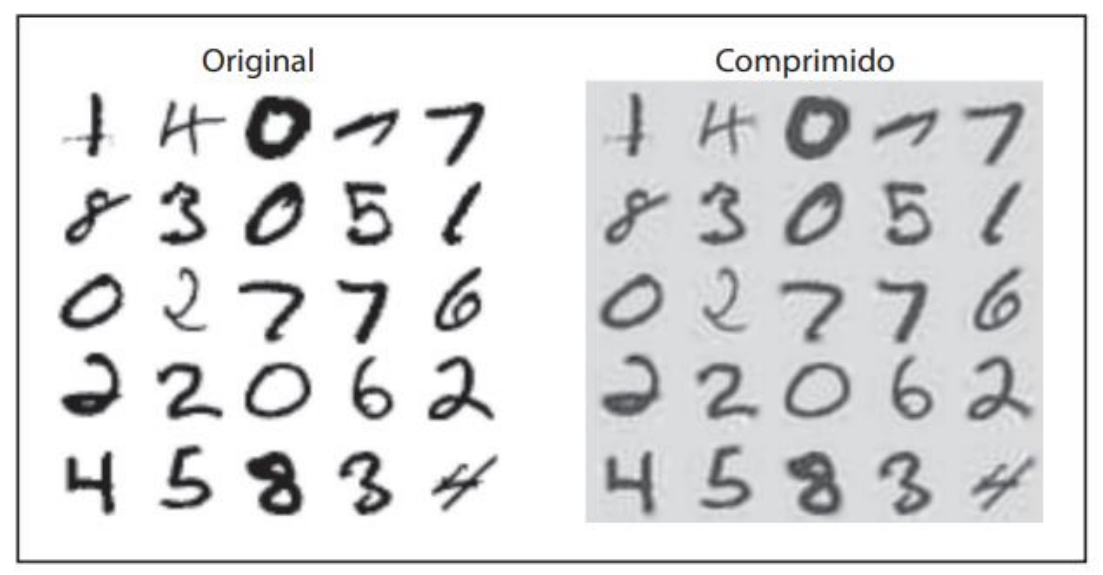
(RUSSELL; NORVIG, 2022. cap. 19-21)

PCA (Análise de Componentes Principais)

- Vantagens
  - Computacionalmente simples e eficiente
  - Fundamentado em conceitos de álgebra linear (vetores e transformações)
- Desvantagens
  - Usa como estimador apenas a variância dos dados em cada dimensão vetorial e considera que todos os atributos são independentes entre si
- Treinamento
  - Computa a variância representada por cada dimensão (atributo) e realiza uma projeção vetorial do conjunto em um número menor de dimensões ao eliminar as dimensões que representam menor variância

PCA – exemplo: redução de dimensionalidade

- Ao aplicar o PCA em um conjunto multidimensional como o da imagem ao lado é possível reduzir consideravelmente o tamanho do conjunto de entrada
- A imagem da direita representa a entrada original que foi comprimida, usando PCA mantendo-se 95% da variância original (5% de perda de informação), e depois descomprimido para a visualização
- É possível verificar que os dígitos ainda são visíveis mesmo que a entrada tenha cerca de 20% do tamanho original



- Original – 784 atributos
- Comprimido – 150 atributos

K-médias

- Vantagens
  - Computacionalmente simples e eficiente
  - Fundamentado em estimadores estatísticos (média)
- Desvantagens
  - Cria apenas grupos de formato circular pois usa apenas a média como estimador
  - Dificuldade na estimativa do parâmetro k (quantidade de grupos)
- Treinamento
  - Computa interativamente k centroides a partir da média de valor dos elementos do grupo a qual está representando. Os elementos são redistribuídos entre os k centroides a cada iteração, usando o critério de proximidade, até que a convergência ocorra (centroides não mudam de valor/posição após uma iteração)

Algoritmo EM (Expectation Maximization)

- Vantagens
  - Mais flexível pois cria grupos de formato elíptico
  - Mais robusto a valores atípicos (outliers)
  - Fundamentado em estimadores estatísticos (média e variância)
- Desvantagens
  - Custoso computacionalmente, especialmente em problemas de alta dimensionalidade (quantidade de atributos de entrada)
  - Assume que os dados de entrada seguem uma distribuição Normal (Gaussiana)
- Treinamento
  - Computa interativamente k centroides a partir da média e da variância de valor dos elementos do grupo a qual está representando. Os elementos são redistribuídos entre os k centroides a cada iteração, usando critério probabilísticos de uma distribuição Normal de dados, até que a convergência ocorra (centroides não mudam de valor/posição após uma iteração)

K-médias e Algoritmo EM – exemplos

- A primeira imagem ao lado compara o agrupamento (k=3) realizado pelo k-médias e pelo algoritmo EM
- A segunda imagem compara a quantização de cores resultante de um algoritmo k-médias para uma mesma foto com diferentes valores para k

K-médias – passos

- Definir o valor de k, esse valor determinará a quantidade de grupos resultante
- Iniciar com k centroides aleatórios
  - Indica-se copiar valores de amostras aleatórias presentes no conjunto de treinamento
- Computar os grupos de cada amostra do conjunto de treinamento considerando o centroide mais próximo
  - Comumente utiliza-se a fórmula da distância Euclidiana *d* entre dois pontos *p* e *q* com *n* atributos (dimensão da entrada)

$$d(p,q) = \sqrt{\sum_{i}^{n}(p_i - q_i)^2}$$

- Computar a média de cada grupo, individualmente por atributo, e usar os valores computados como os novos valores dos centroides
- Repetir os dois últimos passos acima até que os centroides mantenham o mesmo valor

Exercício 6 – k-médias

- Construa um modelo de agrupamento usando o k-médias, com k=3, para o conjunto de entradas abaixo

A1	A2	i = 1	i = 2	i = 3
10	7			
5	5			
7	20			
12	1			
15	2			
16	7			
20	15			
2	10			
5	6			
9	22			

i	Centroide 1	Centroide 2	Centroide 3
1			
2			
3			

Exercício 7 – k-médias

- Construa um modelo de agrupamento usando o k-médias, com k=2, para o conjunto de entradas abaixo

Clima	Temperatura	Humidade	Vento
Chuvoso	Frio	Alta	Intenso
Ensolarado	Agradável	Baixa	Fraco
Nublado	Calor	Normal	Fraco
Nublado	Agradável	Normal	Fraco
Ensolarado	Calor	Normal	Fraco
Nublado	Frio	Normal	Intenso
Chuvoso	Calor	Alta	Intenso
Chuvoso	Agradável	Alta	Intenso
Ensolarado	Agradável	Baixa	Intenso
Nublado	Frio	Baixa	Fraco

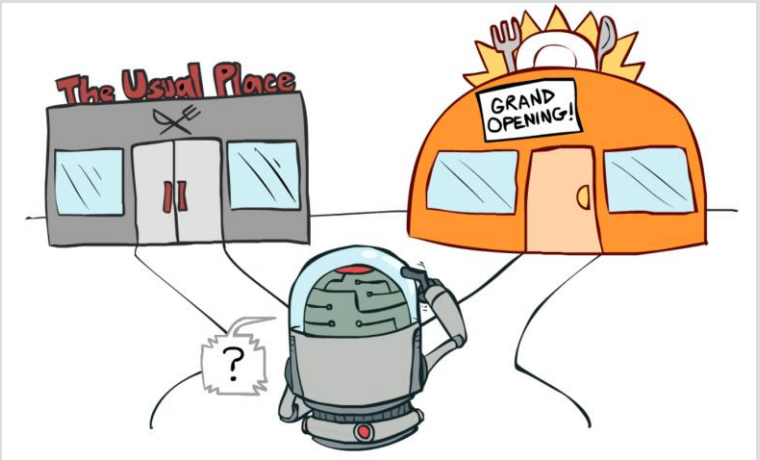
Aprendizado por reforço

Recebe recompensas e punições em ambientes simulados e ajusta seu comportamento de acordo

(RUSSELL; NORVIG, 2022. cap. 19 e 22)

Conceito

- O aprendizado por reforço (*Reinforcement Learning* – RL) foca no aprendizado através de técnicas baseadas em tentativa e erro.
- O treinamento de um agente, em RL, envolve a atualização de um modelo do ambiente, função de utilidade ou política (estratégia) através de recompensas/punições observadas ao interagir com um ambiente simulado de treino.
- Alterna repetidas vezes entre exploração (*exploration*): explora novas possibilidades – e explotação (*exploitation*): reforça e aplica conhecimento já adquirido.



Q-learning – resumo

- Vantagens
  - Algoritmo simples que não precisa de qualquer informação prévia sobre o domínio do problema e não cria um modelo do ambiente (livre de modelo)
  - Fundamentado no conceito de utilidade ou valor da ação
- Desvantagens
  - Pouco eficiente computacionalmente, em especial se as recompensas são muito espaçadas (distantes) no tempo
- Treinamento
  - Alterna entre exploração e explotação em um ambiente simulado repetidas vezes ao mesmo tempo que ajusta constantemente uma função Q (sem hífen ou sem qualidade) que mapeia as recompensas (ou penalidade) acumuladas durante o caminho para cada ação testada

Q-learning – funcionamento

- Aplica uma fórmula baseada em aprendizado temporal que ajusta a função de utilidade conforme as experiência adquiridas e a previsão/expectativa futura
  - $Q(estado, ação) = (1 - \alpha) \times Q(estado, ação) + \alpha \times (recompensa + \gamma \times maxQ(próximoEstado, ações))$
- A taxa de desconto determina a importância que daremos para recompensas futuras (estratégia de longo prazo) frente a uma estratégia mais imediatista
- Ao diminuir a taxa de aprendizado conforme o modelo ganha conhecimento e ao reduzir a taxa de desconto conforme o objetivo final esteja mais próximo torna-se o aprendizado do algoritmo mais eficiente

Variável	Descrição
Q	Função que mapeia o conjunto de estados e suas ações possíveis
α	Taxa de aprendizado, $0 < \alpha \leq 1$
γ	Taxa de desconto, $0 < \gamma \leq 1$
maxQ	Valor máximo da tabela Q em um determinado estado (melhor ação)

Q-learning – passos

- Inicializa-se o mapa (função Q) com zeros para todos os pares de ação/estado possíveis
- Exploração:
  - Selecionar uma ação disponível (aleatoriamente) no estado atual
  - Executar/simular a ação e armazenar a recompensa obtida atualizando o mapa Q
  - Repetir os passos anteriores no novo estado até que o agente atinja o objetivo ou um estado final
  - Inicializar o agente em um estado inicial e repetir todo o processo mais uma vez (até um limite pré-definido de simulações)
- Explotação:
  - Executa os mesmos passos anteriores com a diferença de que as ações são escolhidas a partir do conhecimento já obtido ao invés de aleatoriamente, ou seja, a ação escolhida em cada estado é a ação que maximiza a função Q
- Alterna-se entre as fases exploração e explotação até que o treinamento seja finalizado (agente dominou o problema). Inicialmente a fase de exploração deve ser maior que a de explotação e isso deve ir se invertendo conforme o agente adquire mais conhecimento

Q-learning – exemplo: problema do taxista

- Ambiente de simulação disponível na biblioteca gym (Taxi-v3) do Python
  - https://gymnasium.farama.org/
- O objetivo é transportar passageiros a partir dos pontos {R, G, B, Y} até outro ponto {R, G, B, Y} com o menor custo possível (menor caminho)
- Total de estados (25x5x4 = 500)
  - 25 (grade 5x5) posições para o carro
  - 4 posições para passageiros {R, G, B, Y} +1 para quando este está dentro do carro
  - 4 destinos possíveis {R, G, B, Y}
- Total de ações (6):
  - Sul, norte, leste, oeste, pegar, largar.
- Recompensas:
  - Ação de movimento: -1
  - Ação de pegar ou largar passageiro inválidas: -10
  - Ação de largar um passageiro no destino correto: +20

Bibliotecas em Python

Stable-Baselines3

https://stable-baselines3.readthedocs.io/en/master/

RL_Coach

https://intellabs.github.io/coach/

Tensorforce

https://tensorforce.readthedocs.io/en/latest/

Bibliografia

- BARBETTA, Pedro Alberto; REIS, Marcelo Menezes; BORNIA, Antonio Cezar. **Estatística para cursos de engenharia e informática**. 3. ed. São Paulo: Atlas, 2010. 416 p. ISBN 9788522459940.
- RUSSELL, Stuart J.; NORVIG, Peter. **Inteligência Artificial**: uma abordagem moderna. 4. ed. Rio de Janeiro: GEN LTC, 2022. 1016 p. ISBN 9788595158870.
- SANDERSON, Grant. **Animated Math**: Neural Networks. Estados Unidos da America, 2017. Disponível em: https://www.3blue1brown.com/topics/neural-networks. Acesso em: 26 mar. 2023.
- SANDERSON, Grant. **Animated Math**: Probability. Estados Unidos da America, 2023. Disponível em: https://www.3blue1brown.com/topics/probability. Acesso em: 26 mar. 2023.
- TALBI, El-Ghazali. **Metaheuristics:** from design to implementation. 1. ed. Wiley, 2009. 624 p.