UNIP

UNIVERSIDADE PAULISTA

Inteligência Artificial

Autora: Profa. Vanessa Santos Lessa
Colaboradora: Profa. Larissa Rodrigues Damiani

Professora conteudista: Vanessa Santos Lessa

Doutora em Ciências e Aplicações Geoespaciais pelo Instituto Presbiteriano Mackenzie, mestre em Engenharia Elétrica com ênfase em Inteligência Artificial Aplicada à Automação pelo Centro Universitário da Fundação Educacional Inaciana (FEI) e bacharel em Engenharia da Computação pela Universidade São Judas Tadeu (USJT). Coordenadora do curso de Bacharelado em Ciência da Computação na Universidade Paulista (UNIP) na modalidade Educação a Distância (EaD). Atua há mais de 18 anos em cargos técnicos e gerenciais na área de computação.

Dados Internacionais de Catalogação na Publicação (CIP)

L638i Lessa, Vanessa dos Santos.

Inteligência Artificial / Vanessa dos Santos Lessa. – São Paulo: Editora Sol, 2023.

152 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Inteligência. 2. Algoritmo. 3. Aprendizado. I. Título.

CDU 007.52

U517.49 – 23

© Todos os direitos reservados. Nenhuma parte desta obra pode ser reproduzida ou transmitida por qualquer forma e/ou quaisquer meios (eletrônico, incluindo fotocópia e gravação) ou arquivada em qualquer sistema ou banco de dados sem permissão escrita da Universidade Paulista.

Profa. Sandra Miessa
Reitora

Profa. Dra. Marilia Ancona Lopez
Vice-Reitora de Graduação

Profa. Dra. Marina Ancona Lopez Soligo
Vice-Reitora de Pós-Graduação e Pesquisa

Profa. Dra. Claudia Meucci Andreatini
Vice-Reitora de Administração e Finanças

Prof. Dr. Paschoal Laercio Armonia
Vice-Reitor de Extensão

Prof. Fábio Romeu de Carvalho
Vice-Reitor de Planejamento

Profa. Melânia Dalla Torre
Vice-Reitora das Unidades Universitárias

Profa. Silvia Gomes Miessa
Vice-Reitora de Recursos Humanos e de Pessoal

Profa. Laura Ancona Lee
Vice-Reitora de Relações Internacionais

Prof. Marcus Vinícius Mathias
Vice-Reitor de Assuntos da Comunidade Universitária

UNIP EaD

Profa. Elisabete Brihy
Profa. M. Isabel Cristina Satie Yoshida Tonetto
Prof. M. Ivan Daliberto Frugoli
Prof. Dr. Luiz Felipe Scabar

Material Didático

Comissão editorial:
Profa. Dra. Christiane Mazur Doi
Profa. Dra. Ronilda Ribeiro

Apoio:
Profa. Cláudia Regina Baptista
Profa. M. Deise Alcantara Carreiro
Profa. Ana Paula Tôrres de Novaes Menezes

Projeto gráfico:
Prof. Alexandre Ponzetto

Revisão:
Lucas Ricardi
Kleber Souza

Sumário

Inteligência Artificial

Unidade III

APRESENTAÇÃO

Como futuro cientista da computação, você encontrará desafios envolvendo o desenvolvimento de tecnologias que tenham a capacidade de simular as ações humanas e de pensar de maneira lógica, como chatbot (software capaz de manter uma conversa com um usuário humano em linguagem natural utilizando mensagens, por exemplo), aplicações de gestão, assistente pessoal, mecanismos de segurança, predições, vendas e marketing, ensino etc.

O objetivo desta disciplina é apresentar aos alunos os princípios básicos da inteligência computacional referente às suas diversas áreas, procurando explorar o desenvolvimento de sistemas inteligentes.

Neste livro-texto, você terá a oportunidade de compreender a tecnologia da inteligência artificial, os modelos algorítmicos (e não algorítmicos) básicos para sua implementação, o conceito de sistemas baseados em conhecimento e o espectro de aplicações da inteligência artificial nas diversas áreas profissionais.

Vale acrescentar que este livro-texto é escrito em linguagem simples e direta, como se houvesse uma conversa entre a autora e o leitor. Adicionalmente, são inseridas muitas figuras, que auxiliam no entendimento dos tópicos desenvolvidos. Os itens chamados de observação e de lembrete são oportunidades para que você solucione eventuais dúvidas. Os itens chamados de saiba mais possibilitam que você amplie seus conhecimentos.

INTRODUÇÃO

No mundo dos negócios, as empresas estão sempre armazenando muitos dados. Sejam de fornecedores ou clientes, esses dados devem estar disponíveis para análise, e saber aproveitá-los ao máximo é um diferencial no mercado. Dessa forma, as ferramentas baseadas em inteligência artificial podem coletar e analisar dados com muito mais eficiência do que um ser humano.

A inteligência artificial é a tecnologia que pode executar diferentes funções simultaneamente sem perder a qualidade, ao contrário de um ser humano. Podemos projetar softwares para se comportar de forma inteligente, isto é, semelhante ao comportamento de uma pessoa, utilizando o conceito básico da inteligência, "o aprendizado".

As soluções que utilizam inteligência artificial são desenvolvidas para "aprender" a melhorar o serviço para o qual foram projetadas. Atualmente, podemos identificar a implementação desses softwares em celulares, casas, carros, empresas, hospitais etc. Podemos aplicar inteligência artificial em todos os setores para melhorar o desempenho das tarefas e ajudar os profissionais.

O conteúdo deste livro-texto foi divido em três unidades.

Na unidade I, apresentaremos os conceitos básicos da inteligência artificial e aprofundaremos o conhecimento até a análise de alguns algoritmos.

Na unidade II, abordaremos o pensamento e a representação do conhecimento e do raciocínio.

Na unidade III, aprenderemos sobre aprendizado de máquina supervisionado, não supervisionado e por reforço.

Esperamos que você tenha uma boa leitura e se sinta motivado a ler e conhecer mais sobre a disciplina.

Bons estudos.

Unidade I

1 INTELIGÊNCIA ARTIFICIAL

O nome dado à nossa espécie, de acordo com a classificação taxonômica, é *Homo sapiens*. Esse termo deriva do latim e significa "homem sábio" ou "homem que sabe". Nossa espécie se diferencia das outras espécies do reino animal, pois temos autoconsciência, racionalidade e sapiência. Essas diferenças entre as espécies, principalmente a questão de como pensamos, vem sendo estudada há muitos anos. A forma como entendemos o mundo é um assunto de interesse do campo da inteligência artificial (IA), não apenas o entendimento como também o desenvolvimento de entidades inteligentes.

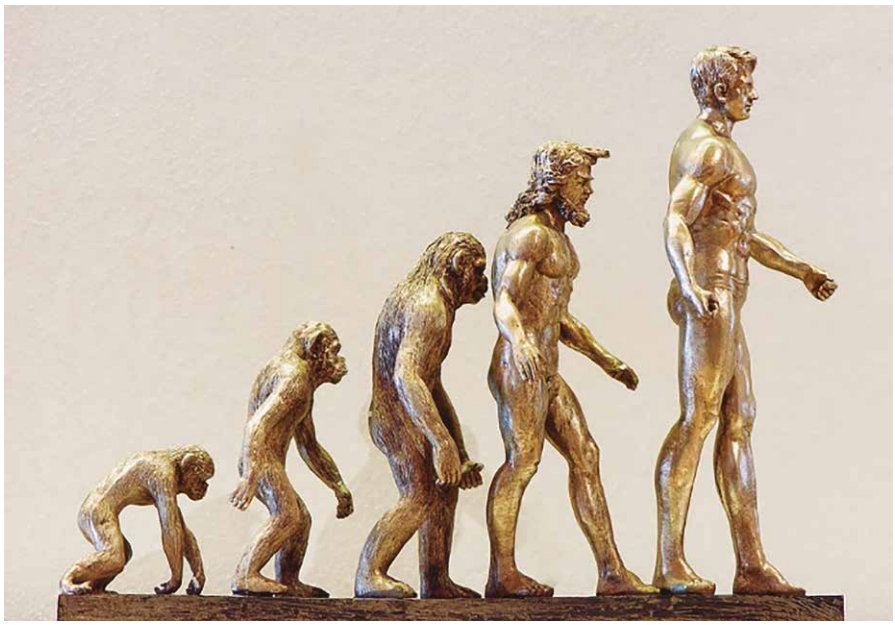


Figura 1

Disponível em: https://cutt.ly/c8EbXat. Acesso em: 3 mar. 2023.

Após a Segunda Guerra Mundial, na Conferência de Dartmouth, em 1956, o termo **inteligência artificial** foi criado por John McCarthy, que foi um cientista da computação conhecido por criar a linguagem de programação Lisp. Ele recebeu o Prêmio Turing em 1972 e a Medalha Nacional de Ciências dos Estados Unidos em 1991.

Saiba mais

A linguagem Lisp foi criada inicialmente para o processamento de dados simbólicos. No artigo a seguir, os autores mostram que é possível utilizar apenas funções matemáticas como estruturas de dados elementares.

MCCARTHY, J. *et al. Lisp 1.5 programmer's manual*. Massachusetts: MIT, 1962.

Vamos começar com duas perguntas importantes:

O que é inteligência?

A inteligência é um conceito complexo que abrange vários aspectos do pensamento e da capacidade cognitiva. Em geral, é entendida como a capacidade de aprender, compreender e aplicar conhecimento e habilidades para resolver problemas e adaptar-se a novas situações. A inteligência também pode ser entendida como a capacidade de raciocinar, planejar, entender ideias abstratas, compreender a linguagem, aprender rapidamente e com a experiência. Diferentes teorias da inteligência propõem distintas definições e medidas da inteligência.

O que é inteligência artificial?

A inteligência artificial (IA) é o ramo da ciência da computação que se dedica ao desenvolvimento de algoritmos e técnicas que permitem que as máquinas simulem a inteligência humana. Isso inclui tarefas como aprendizado automático, processamento de linguagem natural, visão computacional e tomada de decisão. A IA tem aplicações em muitos campos, como medicina, finanças, robótica e automação industrial.

A capacidade de um sistema compreender corretamente dados externos, aprender com esses dados e conseguir utilizar esse aprendizado para realizar algum objetivo especifico define os passos da inteligência artificial (KAPLAN; HAENLEIN, 2019).

A ideia deste primeiro tópico é apresentar os fundamentos e conceitos de IA. Vamos começar com as áreas e subáreas que a fundamentam e alguns de seus algoritmos conhecidos.

1.1 Fundamentos da inteligência artificial

Diversas áreas forneceram ideias, percepções e técnicas para a área da IA. Filosofia, matemática, economia, neurociência, psicologia, engenharia da computação, teoria de controle, cibernética e linguística são algumas ciências que contribuíram com a IA, mas isso não significa que sejam os únicos tópicos com os quais a IA trabalha ou que todas essas disciplinas estão caminhando para a IA como sua conquista final.

Filosofia

As contribuições da filosofia começam com Aristóteles (384–322 a.C.), como podemos observar no quadro a seguir:

Quadro 1

Quem	Contribuição
Aristóteles (384–322 a.C.)	Ele foi o primeiro a formular um conjunto de leis que regem a parte racional da mente precisa. Ele desenvolveu um sistema informal de silogismos para raciocínio ordenado que basicamente permitiu tirar conclusões mecânicas dadas as premissas iniciais
Ramon Llull (1232–1315)	Apresentou a ideia de que o raciocínio útil poderia na realidade ser conduzido por um artefato mecânico
Thomas Hobbes (1588–1679)	Propôs que o raciocínio era semelhante à computação numérica, ou seja, que "efetuamos somas e subtrações em nossos pensamentos silenciosos"
Leonardo da Vinci (1452–1519)	Projetou, mas não construiu, uma calculadora mecânica; reconstruções futuras mostraram que o projeto era funcional
Wilhelm Schickard (1592–1635)	A primeira máquina de calcular conhecida foi construída em torno de 1623 pelo cientista alemão
Blaise Pascal (1623–1662)	Escreveu que "a máquina aritmética produz efeitos que parecem mais próximos ao pensamento que todas as ações dos animais"
Gottfried Wilhelm Leibniz (1646–1716)	Construiu um dispositivo mecânico destinado a efetuar operações sobre conceitos, e não sobre números
Thomas Hobbes (1588–1679)	Com o livro *Leviatã* (1651), sugeriu a ideia de um "animal artificial", argumentando: "Pois o que é o coração, senão uma mola; e os nervos, senão tantas cordas; e as articulações, senão tantas rodas"
René Descartes (1596–1650)	Apresentou a primeira discussão clara da distinção entre mente e matéria, e dos problemas que surgem dessa distinção
Francis Bacon (1561–1626)	O movimento chamado empirismo (acredita que as experiências humanas são as únicas responsáveis pela formação das ideias e conceitos existentes no mundo) foi iniciado a partir de sua obra *Novum organum*
David Hume (1711–1776)	A obra *A treatise of human nature* (1739) propôs aquilo que se conhece hoje como o princípio de indução: as regras gerais são adquiridas pela exposição a associações repetidas entre seus elementos
Rudolf Carnap (1891–1970) Ludwig Wittgenstein (1889–1951) Bertrand Russell (1872–1970)	O famoso Círculo de Viena, liderado por Rudolf Carnap, desenvolveu a doutrina do positivismo lógico. Essa doutrina sustenta que todo conhecimento pode ser caracterizado por teorias lógicas conectadas, em última análise, a sentenças de observação que correspondem a entradas sensoriais; desse modo, o positivismo lógico combina o racionalismo e o empirismo
Rudolf Carnap (1891–1970) Carl Hempel (1905–1997)	Criou a teoria da confirmação que tentava compreender a aquisição do conhecimento através da experiência
Rudolf Carnap (1891–1970)	O livro *The logical structure of the world* (1928) definiu um procedimento computacional explícito para extrair conhecimento de experiências elementares. Provavelmente, foi a primeira teoria da mente como um processo computacional

Adaptado de: Russell e Norvig (2013, p. 6).

Descartes foi um forte defensor do poder da razão para entender o mundo. A filosofia à qual Aristóteles e Leibniz pertenciam é atualmente conhecida como racionalismo. Descartes também foi um defensor do dualismo. Ele disse que existe uma parte da mente humana (ou alma ou espírito) que transcende a natureza e está isenta das leis da física. Os animais, por outro lado, não têm essa natureza dual. Eles podem ser tratados como máquinas. A alternativa ao dualismo é o materialismo. O materialismo diz que as funções do cérebro de acordo com as leis da física constituem a mente. O livre-arbítrio é como os eleitores veem as opções disponíveis.

A conexão entre conhecimento e ação é uma questão vital para a IA porque a inteligência exige ação, bem como raciocínio. Além disso, apenas pela compreensão de como as ações são justificadas podemos compreender como construir um agente cujas ações sejam justificáveis (ou racionais). Aristóteles argumentava, em seu *De motu animalium*, que as ações se justificam por uma conexão lógica entre metas e conhecimento do resultado da ação.

Na obra *Ética a Nicômaco* (1987, p. 1112b), Aristóteles desenvolve esse tópico um pouco mais, sugerindo um algoritmo:

> Não deliberamos sobre os fins, mas sobre os meios. Um médico não delibera sobre se deve ou não curar nem um orador sobre se deve ou não persuadir, [...] Eles dão a finalidade por estabelecida e procuram saber a maneira de alcançá-la; se lhes parece poder ser alcançada por vários meios, procuram saber o mais fácil e o mais eficaz; e se há apenas um meio para alcançá-la, procuram saber como será alcançada por esse meio e por que outro meio alcançar esse primeiro, até chegar ao primeiro princípio, que é o último na ordem de descoberta. [...] e o que vem em último lugar na ordem da análise parece ser o primeiro na ordem da execução. E, se chegarmos a uma impossibilidade, abandonamos a busca; por exemplo, se precisarmos de dinheiro e não for possível consegui-lo; porém, se algo parecer possível, tentaremos realizá-lo.

A filosofia tem debatido sobre o que é inteligência há séculos. Essas discussões forneceram uma base para a definição de inteligência artificial e para a avaliação dos sistemas de IA. A lógica formal foi desenvolvida pela filosofia para estudar o raciocínio lógico e a dedução. Essas técnicas são amplamente utilizadas em sistemas de IA, especialmente naqueles baseados em regras.

Outro debate importante da filosofia é sobre como adquirimos conhecimento e como podemos avaliar a validade desse conhecimento. Essas discussões forneceram uma base para a compreensão da aquisição de conhecimento em sistemas de IA. A filosofia tem debatido sobre questões éticas relacionadas à tecnologia, incluindo a IA. Esses debates ajudam a garantir que os sistemas de IA sejam desenvolvidos e utilizados de maneira ética e responsável.

A filosofia da mente se concentra em entender como a mente funciona e como ela se relaciona com o cérebro. Essa disciplina fornece uma base para a compreensão da inteligência artificial, já que a IA

busca simular a inteligência humana. A epistemologia é a teoria do conhecimento, e ela é importante para a inteligência artificial porque ajuda a compreender como o conhecimento é adquirido, organizado e utilizado por sistemas de IA.

É importante notar que essas contribuições não são lineares e interdependentes, muitas vezes se complementam e se influenciam mutuamente. Além disso, a filosofia continua a desempenhar um papel importante na orientação do desenvolvimento e aplicação da inteligência artificial.

Matemática

Os filósofos apresentaram as ideias mais importantes sobre inteligência artificial, mas o avanço para a ciência formal exigiu uma formalização matemática em três importantes áreas: lógica, computação e probabilidade.

Quadro 2

Quem	Contribuição
Girolamo Cardano (1501–1576)	Utilizou de probabilidade para descrever os possíveis resultados de um jogo de azar
Pierre de Fermat (1601–1665) Blaise Pascal (1623–1662)	Utilizaram a probabilidade para prever o futuro de um jogo de azar inacabado
Thomas Bayes (1702–1761)	Propôs uma regra para atualizar as probabilidades. A regra de Bayes e o campo resultante chamado de análise bayesiana fundamentam o raciocínio de incerteza em sistemas de inteligência artificial
George Boole (1815–1864)	Definiu os detalhes da lógica proposicional ou lógica booleana
Gottlob Frege (1848–1925)	Estendeu a lógica booleana para incluir objetos e relacionamentos, criando a lógica de primeira ordem que usamos hoje
Alfred Tarski (1902–1983)	Introduziu a Teoria Comparativa, que mostra como os objetos se relacionam uns com os outros
Kurt Gödel (1906–1978)	Mostrou que a lógica de primeira ordem de Frege e Russell era eficaz, mas possuía uma falhava em capturar o princípio da indução matemática necessária para caracterizar os números naturais Criou o teorema da incompletude mostrando que toda teoria formal tão forte quanto a aritmética de Peano (a Teoria dos Números Naturais) tem afirmações verdadeiras que são indefiníveis no sentido de que a teoria não tem prova
Alan Turing (1912–1954)	Tentou caracterizar exatamente quais funções são computáveis e quais podem ser computáveis. Na realidade, esse conceito é um tanto problemático, porque o conceito de computação ou procedimento eficiente não pode ser definido formalmente

Quem	Contribuição
Alonzo Church (1903–1995)	Na tese de Church-Turing, mostra que uma máquina de Turing pode calcular qualquer função computável, e essa definição é geralmente aceita como uma definição suficiente. Também mostrou que existem algumas funções que nenhuma máquina de Turing pode calcular
Alan Cobham (1927–2011) Jack Edmonds (1934)	Apontaram para a distinção entre crescimento polinomial e exponencial em complexidade. O crescimento exponencial significa que mesmo casos moderadamente grandes não podem ser resolvidos em um período razoável; dessa forma, devemos decompor o problema geral em subproblemas gerenciáveis, em vez de subproblemas intratáveis
Stephen Cook (1939) Richard Karp (1935)	Apresentaram a teoria da NP-completude, um método para identificar problemas intratáveis. Mostraram que existem grandes classes de combinatória canônica e problemas de inferência que são NP-completos, e que qualquer classe de problemas que for reduzida a uma classe de problemas NP-completos é provavelmente intratável

Adaptado de: Russell e Norvig (2013, p. 7).

Os trabalhos de diversos pesquisadores contribuíram e contribuem para a formalização matemática da ciência.

A álgebra linear é uma área da matemática que trata de vetores e matrizes. É usada para resolver problemas de otimização em sistemas de IA, incluindo o treinamento de redes neurais, conforme veremos adiante.

Já a estatística é uma área da matemática que trata de coleta, análise e interpretação de dados. É usada para avaliar a precisão e a confiabilidade dos sistemas de IA, bem como para selecionar e ajustar modelos.

A Teoria da Computação estuda as propriedades fundamentais dos algoritmos e a complexidade computacional. É usada para projetar algoritmos eficientes para sistemas de IA, incluindo aprendizado de máquina e inteligência artificial baseada em regras.

Por sua vez, a Teoria dos Grafos estuda a estrutura e as propriedades de redes de nós e arestas. É usada para modelar e analisar sistemas complexos, incluindo sistemas de IA.

O cálculo estuda as propriedades das funções e suas derivadas. É usado para otimizar os parâmetros dos modelos de IA e para resolver problemas de otimização não lineares.

Citamos também a Teoria da Informação, que estuda a quantidade, a natureza e a distribuição da informação. É usada para projetar algoritmos de codificação eficientes para sistemas de IA e para medir a complexidade dos problemas.

Podemos relacionar também a Teoria da Decisão, que é uma área da matemática que estuda como as pessoas e as organizações tomam decisões. É usada para desenvolver sistemas de IA que possam tomar decisões racionais e eficientes.

É sempre importante notar que estas contribuições matemáticas não são lineares e interdependentes e que muitas vezes se complementam e se influenciam mutuamente. Além disso, a matemática continua a desempenhar um papel importante no desenvolvimento e aplicação da inteligência artificial.

Economia

Em 1776, o filósofo escocês Adam Smith (1723–1790) publicou *An inquiry into the nature and causes of the wealth of nations* e tratou pela primeira vez a economia como ciência. Apresentou o conceito de que podemos levar em conta que as economias constituem em agentes individuais que maximizam seu próprio bem-estar econômico.

A economia tem contribuído de várias maneiras para o desenvolvimento da IA. Algumas dessas contribuições incluem:

- **Teoria dos Jogos**: estuda como as pessoas e as organizações tomam decisões em situações de incerteza e com vários jogadores. É usada para desenvolver sistemas de IA que possam tomar decisões racionais e eficientes em situações de incerteza e competição.

- **Teoria da Escolha Racional**: estuda como as pessoas e as organizações tomam decisões. É usada para desenvolver sistemas de IA que possam tomar decisões racionais e eficientes.

- **Teoria da Eficiência**: estuda como alocar recursos de maneira eficiente. É usada para desenvolver sistemas de IA que possam alocar recursos de maneira eficiente.

- **Teoria da Informação Econômica**: estuda como a informação afeta as decisões econômicas. É usada para desenvolver sistemas de IA que possam gerenciar e processar grandes volumes de dados.

- **Teoria da Organização**: estuda como as organizações funcionam e como elas podem ser projetadas para maximizar a eficiência. É usada para desenvolver sistemas de IA que possam ser integrados de maneira eficiente em organizações.

- **Microeconomia**: estuda o comportamento dos indivíduos, famílias e empresas em relação a preços e mercados. É usada para desenvolver sistemas de IA que possam prever e entender o comportamento de mercado e assim melhorar as decisões comerciais.

- **Macroeconomia**: estuda a economia global, incluindo o crescimento econômico, inflação, desemprego e políticas econômicas. É usada para desenvolver sistemas de IA que possam prever e entender tendências econômicas e assim melhorar as decisões estratégicas.

Neurociência

A neurociência se dedica ao estudo do sistema nervoso, incluindo o cérebro, a medula espinhal e os nervos periféricos. Ela é uma disciplina interdisciplinar que combina a biologia, a química, a psicologia, a medicina e outras áreas para compreender como o sistema nervoso funciona e como isso afeta o comportamento e a cognição.

A neurociência se divide em várias subdisciplinas, incluindo a neuroanatomia, que estuda a estrutura do sistema nervoso; a neurofisiologia, que estuda como os neurônios e as sinapses funcionam; a neuroquímica, que estuda os neurotransmissores e outras moléculas envolvidas na comunicação nervosa; a neuropsicologia, que estuda como o cérebro afeta o comportamento e a cognição; e a neurobiologia celular, que estuda as células do sistema nervoso e como elas trabalham juntas para processar a informação. Essa ciência também está se desenvolvendo rapidamente em áreas como a neurociência computacional, que usa técnicas de computação para simular o cérebro e estudar como ele processa a informação, e a neurociência clínica, que usa as descobertas da neurociência básica para desenvolver novos tratamentos para doenças do sistema nervoso.

As descobertas da neurociência têm implicações importantes para várias áreas, incluindo a saúde, a educação, a psicologia e a tecnologia. Por exemplo, os neurocientistas estão trabalhando para desenvolver novos tratamentos para doenças do cérebro, como o mal de Alzheimer e a depressão, e para melhorar a compreensão da aprendizagem e da memória. A neurociência também está ajudando a desenvolver novas tecnologias, como os dispositivos de realidade virtual e os implantes cerebrais, que podem ajudar as pessoas com deficiências e melhorar a qualidade de vida.

Os neurônios são as células que compõem o sistema nervoso e são responsáveis por transmitir informações através do corpo. Camillo Golgi (1843–1926) desenvolveu em 1873 uma técnica de coloração que permitiu a observação de neurônios individuais no cérebro. A figura a seguir representa as partes de uma célula nervosa ou neurônio:

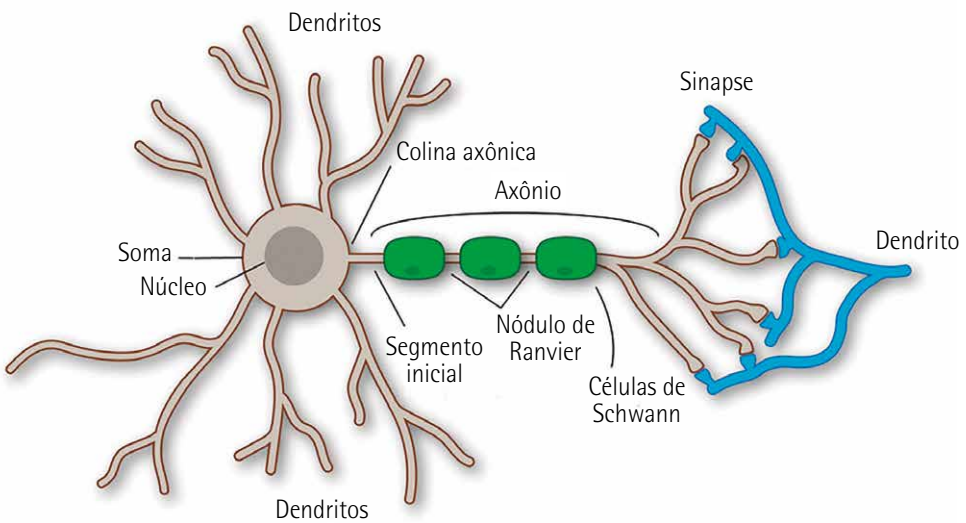


Figura 2 – Partes de uma célula nervosa ou neurônio

Disponível em: https://cutt.ly/W8PxaCB. Acesso em: 6 mar. 2023.

Russell e Norvig (2013, p. 10) descreveram:

> Cada neurônio consiste em um corpo celular ou soma, que contém um núcleo celular. Ramificando-se a partir do corpo celular, há uma série de fibras chamadas dendritos e uma única fibra longa chamada axônio. O axônio se estende por uma longa distância. Em geral, um axônio tem 1 cm de comprimento (100 vezes o diâmetro do corpo celular), mas pode alcançar até 1 metro. Um neurônio faz conexões com 10-100.000 outros neurônios, em junções chamadas sinapses. Os sinais se propagam de um neurônio para outro por meio de uma complicada reação eletroquímica. Os sinais controlam a atividade cerebral em curto prazo e também permitem mudanças a longo prazo na posição e na conectividade dos neurônios. Acredita-se que esses mecanismos formem a base para o aprendizado no cérebro. A maior parte do processamento de informações ocorre no córtex cerebral, a camada exterior do cérebro. A unidade organizacional básica parece ser uma coluna de tecido com aproximadamente 0,5 mm de diâmetro, contendo cerca de 20.000 neurônios e estendendo-se por toda a profundidade do córtex, cerca de 4 mm nos seres humanos.

A neurociência tem contribuído de várias maneiras para o desenvolvimento da IA. Algumas dessas contribuições incluem:

- **Modelos de redes neurais**: a neurociência forneceu a base para o desenvolvimento dos modelos de redes neurais, que são amplamente utilizados em sistemas de IA para problemas de classificação e aprendizado profundo.

- **Compreensão da neuroplasticidade**: a neurociência tem estudado como o cérebro se adapta e aprende, o que é fundamental para desenvolver algoritmos de aprendizado de máquina.

- **Estudos da percepção**: a neurociência tem estudado como o cérebro processa as informações sensoriais, o que é fundamental para desenvolver sistemas de IA capazes de processar dados sensoriais.

- **Estudos da linguagem**: a neurociência tem estudado como o cérebro processa a linguagem, o que é fundamental para desenvolver sistemas de IA capazes de compreender e produzir linguagem natural.

- **Estudos da memória**: a neurociência tem estudado como o cérebro armazena e recupera informações, o que é fundamental para desenvolver sistemas de IA capazes de armazenar e recuperar informações.

- **Estudos do raciocínio**: a neurociência tem estudado como o cérebro raciocina, o que é fundamental para desenvolver sistemas de IA capazes de raciocinar.

- **Estudos da atenção**: a neurociência tem estudado como o cérebro seleciona e filtra informações, o que é fundamental para desenvolver sistemas de IA capazes de selecionar e filtrar informações.

- **Estudos do aprendizado**: a neurociência tem estudado como o cérebro aprende, o que é fundamental para desenvolver sistemas de IA capazes de aprender.

Além disso, os neurocientistas trabalham com matemáticos, engenheiros e psicólogos para desenvolver modelos neurocomputacionais, que buscam entender o funcionamento do cérebro e aplicá-lo para a construção de sistemas de inteligência artificial.

Psicologia

A psicologia é a ciência que estuda o comportamento humano e os processos mentais, incluindo o pensamento e a emoção. Ela se baseia em métodos científicos para compreender como as pessoas pensam, sentem e se comportam, e como esses processos são influenciados por fatores internos e externos.

Existem várias áreas de especialização dentro da psicologia, incluindo: psicologia clínica, que se concentra na compreensão e tratamento de problemas de saúde mental; psicologia cognitiva, que se concentra no estudo do pensamento e da memória; e psicologia social, que se concentra na compreensão da interação social e do comportamento em grupo.

A psicologia tem aplicações em vários campos, incluindo saúde, educação, negócios e justiça criminal. Por exemplo, psicólogos clínicos trabalham com pacientes que têm problemas de saúde mental, enquanto psicólogos organizacionais trabalham com empresas para melhorar a eficiência e a satisfação dos funcionários.

Os psicólogos usam uma variedade de técnicas para coletar dados, incluindo entrevistas, questionários, testes psicológicos e observação. Eles também usam diferentes teorias para explicar os dados coletados, incluindo teorias evolutivas, biológicas e sociais.

A psicologia é uma ciência em constante evolução, e novos achados científicos estão constantemente sendo descobertos, trazendo novas compreensões e possibilidades para ajudar as pessoas a viverem melhores vidas. Trata-se de uma ciência multidisciplinar que se relaciona com outras áreas de conhecimento, como a biologia, a filosofia, a sociologia e a antropologia.

Ela tem contribuído de várias maneiras para o desenvolvimento da IA: por exemplo, a partir do estudo de como as pessoas processam informações sensoriais, o que é fundamental para desenvolver sistemas de IA capazes de processar dados sensoriais, e de como as pessoas processam a linguagem, para desenvolver sistemas de IA capazes de compreender e produzir linguagem natural. Também estuda como as pessoas armazenam e recuperam informações, algo decisivo para desenvolver sistemas de IA capazes de armazenar e recuperar informações, e como as pessoas raciocinam, para desenvolver sistemas de IA capazes de raciocinar. Além disso, analisa como as pessoas selecionam e filtram informações, algo vital para desenvolver sistemas de IA capazes de selecionar e filtrar informações, como as pessoas aprendem, para desenvolver sistemas de IA capazes de aprender, e também o estudo da inteligência, incluindo

como ela se desenvolve, como ela é medida e como ela é relacionada a outras habilidades cognitivas e comportamentais, para desenvolver sistemas de IA que possam simular a inteligência humana. Ainda tem estudado como as pessoas se relacionam umas com as outras, como as expectativas sociais influenciam o comportamento e como as emoções o afetam, algo fundamental para desenvolver sistemas de IA que possam se relacionar de maneira natural com os humanos. Por fim, tem contribuído para a compreensão da interação homem-computador, incluindo como as pessoas se relacionam com os sistemas de IA e como as expectativas afetam a percepção de sistemas de IA.

Engenharia da computação

A engenharia da computação é uma área de estudo multidisciplinar que combina princípios de engenharia, matemática e ciência da computação para projetar, desenvolver, testar e manter sistemas de computação e software. Os engenheiros de computação trabalham em uma variedade de projetos, desde dispositivos móveis até sistemas de computação em nuvem, e desde sistemas embarcados até inteligência artificial.

Uma das principais áreas de atuação da engenharia da computação é o desenvolvimento de software. Os engenheiros de computação usam princípios de programação para criar aplicativos e sistemas de software que podem ser usados em vários campos, como negócios, saúde, educação e jogos. Além disso, eles trabalham em equipe com outros profissionais, como gerentes de projetos e designers de usuários, para garantir que o software seja fácil de usar e atenda às necessidades dos usuários.

Outra área importante da engenharia da computação é o projeto e a construção de sistemas de computação. Isso inclui a criação de arquiteturas de computadores, desde circuitos e dispositivos até sistemas operacionais e redes. Os engenheiros de computação também trabalham para garantir que os sistemas de computação sejam seguros, escaláveis e confiáveis.

A engenharia da computação também está envolvida com outras áreas de tecnologia emergentes, como inteligência artificial, robótica, aprendizado de máquina, visão computacional, automação e automação industrial. Ela tem contribuído de várias maneiras para o desenvolvimento da IA. Algumas dessas contribuições incluem:

- **Desenvolvimento de algoritmos**: desenvolvendo algoritmos para o aprendizado de máquina, o reconhecimento de padrões, a otimização e outras tarefas fundamentais para a IA.

- **Desenvolvimento de arquiteturas**: desenvolvendo arquiteturas para sistemas de IA, incluindo arquiteturas de redes neurais, arquiteturas baseadas em regras e arquiteturas híbridas.

- **Desenvolvimento de ferramentas**: desenvolvendo ferramentas para ajudar a desenvolver e implementar sistemas de IA, incluindo ferramentas para o treinamento e a validação de modelos, ferramentas para a gestão de dados e ferramentas para a otimização de recursos computacionais.

- **Desenvolvimento de sistemas**: desenvolvendo sistemas de IA, incluindo sistemas de aprendizado de máquina, sistemas baseados em regras, sistemas de processamento de linguagem natural e sistemas de visão computacional.

- **Desenvolvimento de infraestrutura**: desenvolvendo infraestrutura para suportar sistemas de IA, incluindo infraestrutura de nuvem, infraestrutura de hardware especializado, infraestrutura de segurança e infraestrutura de gerenciamento de dados.

- **Otimização de recursos**: otimizando recursos necessários para o desenvolvimento de sistemas de IA, incluindo tempo de processamento, uso de memória e uso de banda de dados.

- **Integração com outras tecnologias**: desenvolvendo meios para integrar sistemas de IA com outras tecnologias, como robótica, internet das coisas (IoT), automação e inteligência artificial distribuída.

Teoria de Controle e cibernética

A Teoria de Controle e a cibernética compõem um campo de estudo que se concentra no desenvolvimento de modelos matemáticos e algoritmos para controlar sistemas dinâmicos. Isso inclui sistemas mecânicos, elétricos, biológicos e de informação. São usadas para projetar sistemas capazes de manter as condições desejadas, mesmo quando há variações imprevisíveis nas condições do ambiente.

A cibernética é uma área de estudo que se concentra nos processos de comunicação e controle nos sistemas vivos e não vivos. É usada para projetar sistemas capazes de processar informações e tomar decisões baseadas nessas informações. A Teoria de Controle e a cibernética são interdependentes, já que a primeira fornece os fundamentos matemáticos para a cibernética, enquanto a segunda fornece os princípios de comunicação e controle para a Teoria de Controle.

A Teoria de Controle e a cibernética têm contribuído de várias maneiras para o desenvolvimento da IA. Algumas dessas contribuições incluem:

- **Conceitos de feedback e controle**: a Teoria de Controle forneceu conceitos fundamentais de feedback e controle, que são amplamente utilizados em sistemas de IA para ajustar o comportamento dos sistemas de acordo com os resultados desejados.

- **Modelos de sistemas dinâmicos**: a Teoria de Controle tem desenvolvido modelos matemáticos de sistemas dinâmicos, que são amplamente utilizados em sistemas de IA para prever e controlar o comportamento de sistemas complexos.

- **Controladores baseados em modelos**: a Teoria de Controle tem desenvolvido controladores baseados em modelos, que são amplamente utilizados em sistemas de IA para controlar o comportamento de sistemas automatizados.

- **Teoria de sistemas inteligentes**: a cibernética forneceu a base para a Teoria de Sistemas Inteligentes, que estuda como criar sistemas que possam aprender, se adaptar e se auto-organizar. Isso é fundamental para desenvolver sistemas de IA que possam aprender e se adaptar.

- **Teoria de sistemas autônomos**: a cibernética forneceu a base para a Teoria de Sistemas Autônomos, que estuda como criar sistemas que possam tomar decisões e agir de forma independente. Isso é fundamental para desenvolver sistemas de IA que possam tomar decisões e agir de forma autônoma.

- **Teoria de sistemas adaptativos**: a cibernética forneceu a base para a Teoria de Sistemas Adaptativos, que estuda como criar sistemas que possam se adaptar a mudanças no ambiente. Isso é fundamental para desenvolver sistemas de IA que possam se adaptar a mudanças no ambiente.

- **Teoria de sistemas inteligentes distribuídos**: a cibernética forneceu a base para a Teoria de Sistemas Inteligentes Distribuídos, que estuda como criar sistemas distribuídos que possam trabalhar juntos de forma inteligente. Isso é fundamental para desenvolver sistemas de IA distribuídos que possam trabalhar juntos de forma inteligente.

Linguística

A linguística é o estudo científico da língua e da linguagem humana. É uma disciplina interdisciplinar que se relaciona com outras áreas como a psicologia, a antropologia, a filosofia e a informática. A linguística estuda as estruturas, as regras e os princípios que governam a língua, incluindo a fonética, a fonologia, a morfologia, a sintaxe, a semântica e a pragmática.

A fonética é o estudo dos sons da língua, incluindo a produção, a percepção e a transcrição fonética. A fonologia é o estudo das regras que governam a combinação e a distribuição dos sons da língua. A morfologia é o estudo das palavras e das suas estruturas, incluindo a formação de palavras, a inflexão e a derivação. A sintaxe é o estudo das regras que governam a ordem das palavras e a construção das frases. A semântica é o estudo do significado das palavras, das frases e dos textos. A pragmática é o estudo das regras que governam o uso da língua em contextos sociais e comunicativos.

A linguística também se divide em várias subdisciplinas, como a linguística histórica, que estuda as mudanças e a evolução das línguas ao longo do tempo, e a linguística comparativa, que compara diferentes línguas para entender suas relações genealógicas e suas estruturas comuns.

A linguística descritiva estuda as línguas como sistemas formais e independentes, e a linguística aplicada é utilizada em diferentes campos, como a educação, a tradução, a inteligência artificial e a comunicação.

A linguística tem contribuído de várias maneiras para o desenvolvimento da IA. Algumas dessas contribuições incluem:

- **Compreensão da linguagem natural**: estudando como as línguas funcionam, incluindo sua estrutura, sintaxe e semântica, o que é fundamental para desenvolver sistemas de IA capazes de compreender e produzir linguagem natural.

- **Análise de sentimentos**: estudando como os sentimentos são expressos na linguagem, o que é fundamental para desenvolver sistemas de IA capazes de identificar e analisar sentimentos em textos.

- **Técnicas de processamento de linguagem natural**: desenvolvendo técnicas de processamento de linguagem natural, incluindo análise sintática, análise semântica e geração de texto, que são amplamente utilizadas em sistemas de IA para problemas de processamento de linguagem natural.

- **Análise de discurso**: estudando como as pessoas se comunicam, incluindo como as estruturas de discurso e os contextos afetam a interpretação, o que é fundamental para desenvolver sistemas de IA capazes de compreender e produzir textos de forma natural.

- **Estudos de pragmática**: estudando como as pessoas usam a linguagem para comunicar intenções, inferir significado implícito e lidar com ambiguidade, o que é fundamental para desenvolver sistemas de IA capazes de compreender e produzir linguagem natural.

- **Estudos de semântica**: estudando como as pessoas usam a linguagem para comunicar significado, incluindo como as palavras e as frases são relacionadas e como as palavras são relacionadas com o mundo, o que é fundamental para desenvolver sistemas de IA capazes de compreender e produzir linguagem natural.

- **Estudos de sintaxe**: estudando como as pessoas usam a linguagem para comunicar significado, incluindo como as palavras e as frases são relacionadas e como as palavras são relacionadas com o mundo, o que é fundamental para desenvolver sistemas de IA capazes de compreender e produzir linguagem natural.

1.2 A evolução da inteligência artificial

As origens da inteligência artificial (1943–1955)

Warren McCulloch e Walter Pitts publicaram em 1943 um artigo intitulado "A lógica das células neurais" no qual propuseram uma teoria da computação baseada em células neurais. Esse artigo é considerado um marco histórico na área da inteligência artificial e neurociência, pois foi um dos primeiros trabalhos a propor a ideia de que as células neurais podem ser usadas como elementos básicos de computação.

Na teoria proposta por McCulloch e Pitts, as células neurais são vistas como unidades lógicas simples que podem ser conectadas entre si para formar circuitos neurais mais complexos. Eles sugeriram que esses circuitos neurais podem ser usados para realizar operações lógicas, como a negação, a conjunção e a disjunção, e que essas operações podem ser usadas para construir sistemas computacionais mais avançados.

Além de propor uma nova abordagem para a computação, o trabalho de McCulloch e Pitts teve um impacto significativo na neurociência. Eles propuseram que as células neurais podem ser usadas como unidades lógicas básicas para explicar o funcionamento do cérebro, e essa ideia foi desenvolvida posteriormente por outros pesquisadores na área da neurociência computacional.

Em 1950, o matemático americano John McCarthy acreditava ser possível programar computadores para serem inteligentes e propôs o termo **inteligência artificial** em seu artigo "Programs with common sense", o qual marca o início oficial da IA como campo de estudo.

O teste de Turing foi proposto por Alan Turing em 1950 com o objetivo de avaliar a inteligência de uma máquina. Ele é considerado um marco importante na história da inteligência artificial e é ainda hoje um tópico de debate. O teste consiste em uma conversa entre um juiz humano e dois participantes, um humano e uma máquina, que estão escondidos da vista do juiz. Se o juiz não consegue distinguir qual dos participantes é a máquina, ela é considerada inteligente. A ideia por trás do teste é que se uma máquina consegue simular a inteligência humana de forma tão eficaz que é indistinguível de um ser humano, então ela é inteligente. No entanto, existem críticas a esse teste, como o fato de que ele se concentra apenas na capacidade de uma máquina de simular a inteligência humana, e não na sua capacidade de pensar de forma autônoma.

Em 1951, o engenheiro americano Frank Rosenblatt propôs o modelo perceptron como parte de sua tese de doutorado na Cornell Aeronautical Laboratory. As pesquisas de Walter Pitts e Warren McCulloch, sobre um modelo matemático para um neurônio artificial baseado no funcionamento do neurônio humano, foram a base para a pesquisa de Frank Rosenblatt. Somente em 1957 o modelo perceptron nasceu, com o desenvolvimento do modelo em um computador da época, chamado de Mark I Perceptron Machine.

Observação

O perceptron é um algoritmo de aprendizado de máquina supervisionado baseado em redes neurais de camada única, utilizado para classificação binária e capaz de resolver problemas linearmente separáveis, ou seja, problemas em que é possível separar as duas classes de exemplos por uma reta ou hiperplano, mas não é capaz de resolver problemas não linearmente separáveis. Veremos mais detalhes nos próximos tópicos deste livro-texto.

Em 1952, o matemático americano Nathaniel Rochester e seus colegas da IBM desenvolveram um jogo de xadrez eletrônico, que foi o primeiro programa de computador a jogar xadrez: o Los Alamos Chess. Esse programa foi baseado em regras simples e não tinha capacidade de aprendizado ou raciocínio.

Em 1955, o engenheiro americano Herbert Simon e o matemático americano Allen Newell desenvolvem o Logic Theorist, o primeiro programa de computador capaz de resolver problemas matemáticos. Esse programa foi baseado em lógica formal e demonstrou a capacidade dos computadores para realizar tarefas intelectuais.

Nos anos entre 1943 e 1955, os fundamentos da inteligência artificial foram estabelecidos com a proposta do teste de Turing, a proposta do termo **inteligência artificial** e o início do desenvolvimento de algoritmos de aprendizado de máquina como o perceptron. Também foram desenvolvidos os primeiros programas de computador com capacidade de jogar xadrez e resolver problemas matemáticos, mostrando a capacidade deles de realizar tarefas intelectuais.

Um ano importante para a inteligência artificial (1956)

Em 1956, ocorreu o primeiro workshop de IA na Dartmouth College, nos Estados Unidos. Esse workshop foi organizado por John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon e é considerado o marco inicial do campo da IA moderna. Durante esse workshop, os participantes discutiram a possibilidade de criar sistemas de computação capazes de pensar e aprender como os humanos.

Nesse ano, também foi desenvolvido o programa de computador General Problem Solver (GPS) por Herbert Simon, J. C. Shaw e Allen Newell. Esse programa foi projetado para encontrar soluções para problemas gerais, utilizando técnicas de busca e raciocínio lógico. O GPS foi um dos primeiros programas de computador a utilizar a inteligência artificial para resolver problemas.

Ainda em 1956, o engenheiro americano Arthur Samuel desenvolveu um programa de computador capaz de jogar dama. Este programa foi baseado em algoritmos de aprendizado de máquina e foi capaz de jogar dama melhor do que a maioria dos humanos. O programa foi um dos primeiros exemplos de como a IA poderia ser usada para jogos. Nesse ano, o matemático britânico Christopher Strachey desenvolveu outro programa capaz de jogar damas, o Checkers-Player. Ele usou busca heurística para encontrar a melhor jogada, com uma estratégia baseada em jogadas já realizadas e análise de estatísticas de jogadas.

Observação

Heurística é um processo de tomada de decisão baseado em regras simplificadas ou intuições. É um método para resolver problemas que busca uma solução rápida e eficiente, mesmo que não seja necessariamente a solução ideal ou ótima. As heurísticas são frequentemente usadas em IA como um meio de encontrar soluções aproximadas para problemas complexos em tempo hábil. O objetivo da heurística é encontrar uma solução aceitável com base em informações limitadas, sem precisar considerar todas as possibilidades. Veremos mais detalhes nos próximos tópicos deste livro-texto.

Além disso, nesse ano, o matemático americano John McCarthy propôs o uso da programação lógica para o desenvolvimento de sistemas de inteligência artificial. Ele acreditava que a lógica formal poderia ser usada para modelar o raciocínio humano e criar sistemas capazes de raciocinar e tomar decisões.

O ano de 1956 foi importante para o desenvolvimento da inteligência artificial, com o primeiro workshop de IA, a criação de programas de computador capazes de jogar dama e resolver problemas

gerais, e a proposta de usar programação lógica para o desenvolvimento de sistemas de inteligência artificial. Esses avanços foram importantes para estabelecer a IA como um campo de estudo e começar a desenvolver sistemas capazes de pensar e aprender como os humanos.

As expectativas (1952–1969)

A evolução da inteligência artificial de 1952 até 1969 foi marcada por avanços significativos em vários campos, como aprendizado de máquina, processamento de linguagem natural, visão computacional e sistemas especialistas.

Nathaniel Rochester foi um cientista da computação americano que trabalhou no IBM Watson Research Center na década de 1950. Ele é conhecido, conforme mencionamos anteriormente, por ter participado do desenvolvimento do primeiro jogo de xadrez para computador, chamado Los Alamos Chess. O jogo foi desenvolvido em 1950-1952 como parte do projeto Maniac (Mathematical Analyzer, Numerical Integrator, and Computer) na Los Alamos National Laboratory. O jogo era limitado em sua capacidade de jogar xadrez, mas representou um avanço significativo na compreensão da capacidade de computadores para praticar jogos complexos. A contribuição de Rochester para o desenvolvimento do jogo de xadrez para computador foi importante para o avanço da inteligência artificial e da pesquisa em jogos.

Em 1957, o engenheiro americano Frank Rosenblatt, baseando-se no modelo criado em 1951, cria o perceptron, uma máquina de aprendizado simples baseada em redes neurais. Esse foi o primeiro algoritmo de aprendizado de máquina a ser desenvolvido e permitiu que as redes neurais fossem utilizadas para classificação de dados.

Em 1959, o matemático americano Arthur Samuel publicou um artigo sobre a criação de programas de computador capazes de aprender. Esse artigo é considerado um marco na história da IA, pois introduziu a noção de "aprendizado por meio da experiência" como uma forma de desenvolver sistemas de inteligência artificial.

Saiba mais

Para saber mais sobre programas de computador capazes de aprender, recomenda-se a leitura do artigo a seguir. Nele, o autor descreveu o desenvolvimento de um programa de computador para jogar xadrez, que aprendia com a experiência e melhorava suas habilidades ao jogar contra si mesmo. O trabalho de Samuel foi um dos primeiros exemplos de aprendizado de máquina supervisionado, em que o algoritmo é treinado em dados rotulados e melhora suas previsões com base na experiência adquirida. Esse artigo foi um passo importante na compreensão da capacidade de computadores para aprender com a experiência e foi uma contribuição fundamental para o desenvolvimento da inteligência artificial como campo de estudo.

SAMUEL, A. Some studies in machine learning using the game of checkers. *IBM Journal*, v. 3, n. 3. Jul. 1959.

Em 1962, o matemático americano John McCarthy desenvolveu o sistema de programação lógica Lisp, que é amplamente utilizado até hoje em aplicações de IA. Ele é considerado a primeira linguagem de programação de alto nível para inteligência artificial e foi usado para desenvolver muitos dos primeiros sistemas de IA.

Em 1969, o cientista da computação americano Joseph Weizenbaum desenvolveu o programa de computador Eliza, que era capaz de simular uma conversa humana. Eliza foi baseado em técnicas de processamento de linguagem natural e foi um dos primeiros exemplos de como a IA poderia ser usada para simular a comunicação humana.

Outro importante trabalho neste período foi o desenvolvimento de sistemas especialistas por Mycin, um sistema de diagnóstico médico desenvolvido por Edward Shortliffe na Stanford University. Ele usou técnicas de inferência baseadas em regras para diagnosticar doenças infecciosas e foi considerado um dos primeiros sistemas especialistas de sucesso.

Além disso, nesse período, foi desenvolvida a primeira rede neural artificial para reconhecimento de caracteres. A rede foi treinada para reconhecer caracteres manuscritos e foi uma das primeiras aplicações de aprendizado de máquina.

Uma dose de realidade (1966–1973)

A evolução da IA de 1966 até 1973 foi marcada por avanços significativos na área da inteligência artificial conectada com a realidade. Durante esse período, os pesquisadores começaram a se concentrar em como a IA poderia ser aplicada para resolver problemas reais e lidar com dados do mundo real.

Em 1967, o engenheiro americano Ivan Sutherland desenvolveu o sistema Sword of Damocles, que era uma das primeiras demonstrações de realidade virtual. A realidade virtual é um ambiente digital que imita ou simula uma experiência real, para fornecer uma sensação de presença em um mundo virtual. É criada por meio de tecnologias como óculos VR, dispositivos de rastreamento e software de renderização. A realidade virtual é amplamente utilizada para jogos, entretenimento, treinamento militar e simulações empresariais. O sistema criado por Ivan Sutherland usava um visor de tela de olho único para exibir imagens tridimensionais e permitir que o usuário interagisse com elas.

Em 1968, o cientista da computação americano Gordon Earle Moore publicou o artigo "Cramming more components onto integrated circuits", que previu a evolução dos processadores e o aumento da capacidade de processamento. Isso foi chamado de Lei de Moore.

Em 1971, o cientista da computação americano Edward Feigenbaum e o engenheiro americano Nils Nilsson desenvolveram o sistema Dendral, um sistema especialista em química. Este foi um dos primeiros sistemas especialistas a ser desenvolvido e permitiu que os químicos identificassem compostos orgânicos desconhecidos.

Em 1972, o engenheiro americano Victor Scheinman desenvolveu o robô Stanford Arm, que era capaz de manipular objetos no mundo real. Este foi um dos primeiros robôs a ser desenvolvido e permitiu que os pesquisadores estudassem como a IA poderia ser usada para controlar robôs e outros dispositivos mecânicos.

Em 1973, o engenheiro americano Robert Kahn e o cientista da computação americano Vint Cerf desenvolveram o protocolo TCP/IP, que é a base da internet. A capacidade de conectar computadores de forma descentralizada permitiu que os pesquisadores compartilhassem dados e trabalhassem juntos em projetos de IA.

A evolução da inteligência artificial de 1966 a 1973 foi marcada por avanços significativos na área da inteligência artificial conectada com a realidade, incluindo a realidade virtual, a robótica e outros.

Sistemas baseados em conhecimento (1969–1979)

A evolução da inteligência artificial de 1969 a 1979 foi marcada por avanços significativos na área dos sistemas baseados em conhecimento. Durante esse período, os pesquisadores começaram a se concentrar em como a IA poderia ser usada para representar e manipular o conhecimento humano.

Em 1969, o americano Joseph Weizenbaum, professor de ciência da computação no Massachusetts Institute of Technology (MIT), desenvolveu o programa de computador Eliza, que era capaz de simular uma conversa humana. O programa foi projetado para simular uma terapeuta de psicoterapia, baseado em respostas padrão que imitam as respostas de um terapeuta humano. Quando o usuário digita uma frase, Eliza usa um conjunto de regras simples para reordenar e reformular a frase como uma pergunta, deixando a impressão de que a resposta está sendo fornecida por um ser humano. Eliza foi bem recebido como um exemplo de como a tecnologia poderia ser usada para simular uma conversa humana. No entanto, Weizenbaum também alertou sobre os perigos da automação excessiva e a necessidade

de manter o controle humano sobre a inteligência artificial. Apesar de ser uma tecnologia antiga, o programa Eliza ainda é estudado e referenciado como uma das primeiras tentativas de criar inteligência artificial por meio de conversação humana.

Em 1971, o cientista da computação americano Edward Feigenbaum e o engenheiro americano Nils Nilsson desenvolveram o sistema Dendral, um sistema especialista em química. O programa foi projetado para ajudar a identificar a estrutura molecular de compostos químicos complexos. Dendral usa uma combinação de algoritmos de aprendizado de máquina e técnicas de inferência baseadas em conhecimento para determinar a estrutura molecular de compostos. O programa pode processar informações de fontes como espectrometria de massa e análise de ressonância magnética nuclear para determinar a estrutura molecular de compostos desconhecidos. O programa Dendral foi bem-sucedido em identificar a estrutura molecular de compostos e foi amplamente utilizado na comunidade científica para ajudar na identificação de novos compostos. Além disso, foi importante para o desenvolvimento da inteligência artificial e para a compreensão da combinação de algoritmos de aprendizado de máquina e técnicas baseadas em conhecimento.

Em 1972, o engenheiro americano e professor universitário Victor Scheinman desenvolveu o robô Stanford Arm, que era capaz de manipular objetos no mundo real. Ele foi projetado como uma plataforma de pesquisa para aprimorar a compreensão da robótica e da inteligência artificial. O robô Stanford Arm foi desenvolvido usando tecnologias avançadas de controle de motores e sensores, incluindo sensores de posição, velocidade e força. Ele tinha seis graus de liberdade, permitindo que o braço fizesse movimentos complexos e precisos. O robô foi programado usando uma linguagem de programação baseada em símbolos, que permitia que o usuário especificasse as tarefas a serem realizadas pelo braço robótico. Além disso, o robô foi projetado para ser capaz de aprender de suas experiências, permitindo que aprimorasse suas habilidades com o tempo.

Em 1974, o cientista da computação americano Terry Winograd desenvolveu o sistema Shrdlu, que era capaz de entender e responder a comandos de linguagem natural.

Em 1975, o cientista da computação americano Roger Schank desenvolveu o sistema Sam, um sistema especialista em compreensão de histórias.

Em 1977, o cientista da computação americano Patrick Henry Winston desenvolveu o sistema Lem, que era capaz de entender e responder a comandos de linguagem natural.

Em 1978, o cientista da computação americano Bruce Buchanan desenvolveu o Mycin, um sistema especialista em diagnóstico médico. Ele foi um dos primeiros sistemas especialistas a ser desenvolvido e permitiu que os médicos identificassem doenças com base em sintomas e dados de teste. O Mycin foi capaz de explicar suas conclusões e era baseado em regras de inferência. Isso foi um avanço significativo na IA, pois possibilitou que os sistemas especialistas fossem mais fáceis de entender e usar.

Em 1979, o cientista da computação americano John Laird desenvolveu o sistema Soar, um sistema baseado em conhecimento que era capaz de entender e responder a comandos de linguagem natural. Ele foi um dos primeiros sistemas a usar técnicas de compreensão de linguagem natural e permitiu que

os pesquisadores estudassem como a IA poderia ser usada para compreender e responder à linguagem humana. O Soar também introduziu a ideia de usar metacognição para ajudar a IA a aprender e tomar decisões. Isso foi um avanço importante na IA, pois permitiu que os sistemas baseados em conhecimento fossem mais flexíveis e adaptáveis.

A inteligência artificial na indústria (de 1980 até a atualidade)

A evolução da IA na indústria desde 1980 tem sido marcada por avanços significativos em várias áreas, incluindo reconhecimento de fala e visão, aprendizado de máquina, robótica e inteligência artificial distribuída.

A partir dos anos 1980, a IA começou a ser incorporada em vários sistemas de automação industrial e robótica, permitindo a automação de tarefas complexas e aumentando a eficiência e precisão desses sistemas. Além disso, a IA começou a ser usada em sistemas de diagnóstico e previsão, como sistemas de manutenção preditiva em indústrias de petróleo e gás e sistemas de previsão de falhas em indústrias aeroespaciais.

A partir dos anos 1990, o aprendizado de máquina se tornou uma área crescente de interesse, e as técnicas de aprendizado supervisionado e não supervisionado começaram a ser incorporadas em vários sistemas de IA na indústria. Isso permitiu que os sistemas de IA fossem mais flexíveis e capazes de se adaptar às mudanças no ambiente. Além disso, a IA começou a ser usada em sistemas de análise de dados, como sistemas de mineração de dados e análise preditiva, permitindo que as empresas tomassem decisões baseadas em dados e melhorassem a eficiência de seus processos.

A partir dos anos 2000, a visão computacional se tornou uma área importante de pesquisa e desenvolvimento, permitindo que os sistemas de IA processassem e interpretassem imagens e vídeos. Isso permitiu a criação de sistemas de IA capazes de realizar tarefas visuais, como reconhecimento de objetos e análise de vídeo, sendo utilizado em diversas aplicações como na segurança, manufatura, vigilância e outros. Além disso, o processamento de linguagem natural se tornou uma área importante de pesquisa, permitindo que os sistemas de IA entendessem e processassem linguagem natural.

Desde 2010, a IA tem continuado a evoluir rapidamente e tem se tornado cada vez mais presente em diferentes setores industriais. Alguns exemplos incluem:

- **Manufatura**: otimização de processos de produção, controle de qualidade, previsão de falhas em máquinas.
- **Logística**: planejamento de rotas, otimização de armazenamento e gestão de estoques.
- **Saúde**: diagnósticos médicos, monitoramento de pacientes e pesquisa de novos tratamentos.
- **Energia**: otimização de rede elétrica, previsão de demanda de energia e gerenciamento de fontes renováveis.

- **Varejo**: personalização de recomendações de produtos, análise de dados de comportamento do cliente e previsão de tendências de vendas.

- **Seguros**: análise de risco, previsão de sinistros e automação de processos de seguro.

Esses são apenas alguns exemplos de como a IA pode ser utilizada na indústria, há muitas outras áreas em que ela pode ser aplicada com sucesso.

O retorno das redes neurais (de 1986 até a atualidade)

As redes neurais têm tido um grande impacto na inteligência artificial, especialmente nos últimos anos. Algumas das aplicações mais conhecidas das redes neurais incluem sistemas de reconhecimento de fala, tradução automática e reconhecimento de imagens. Além disso, as redes neurais profundas têm sido usadas para alcançar desempenhos recordes em tarefas de aprendizado supervisionado.

A inteligência artificial baseada em redes neurais teve um grande avanço desde 1986, com o surgimento de algoritmos de aprendizado profundo. Isso permitiu que as redes neurais fossem treinadas com grandes conjuntos de dados, o que melhorou significativamente a precisão das previsões. Além disso, o aumento da capacidade de processamento e a disponibilidade de grandes conjuntos de dados permitiram que as redes neurais fossem escaladas para problemas cada vez mais complexos. Atualmente, as redes neurais profundas são amplamente utilizadas em aplicações como visão computacional, processamento de linguagem natural e jogos.

A IA se torna uma ciência (de 1987 até a atualidade)

Desde 1987, a IA tem evoluído rapidamente e se tornou uma ciência cada vez mais importante. Em 1987, ocorreu o chamado Winter of AI, quando o investimento em IA caiu drasticamente, devido ao fracasso em alcançar algumas das metas ambiciosas que haviam sido estabelecidas para a tecnologia. No entanto, desde então, a IA tem se recuperado e experimentado um crescimento constante.

Uma das principais razões para esse crescimento é o aumento do poder de computação, que permitiu aos pesquisadores treinar redes neurais mais complexas e realizar experimentos em grande escala. Além disso, o avanço da tecnologia de aprendizado de máquina, como o aprendizado profundo, permitiu que a IA alcançasse desempenhos surpreendentes em tarefas como reconhecimento de fala e visão computacional.

Outra razão importante para o crescimento da IA é o aumento da disponibilidade de dados. O aumento da quantidade de dados disponíveis para treinar modelos de IA tem sido um fator crítico para o sucesso das redes neurais, especialmente as redes neurais profundas. Além disso, a popularidade crescente de plataformas de aprendizado de máquina como o TensorFlow e o PyTorch tem permitido que os desenvolvedores e cientistas de dados treinem e implementem modelos de IA mais facilmente.

A IA tem evoluído rapidamente e se tornou uma ciência cada vez mais importante. O aumento do poder de computação, o avanço da tecnologia de aprendizado de máquina e a disponibilidade crescente

de dados foram algumas das principais razões para este crescimento. Hoje, a IA é amplamente utilizada em várias indústrias e tem o potencial de transformar a forma como vivemos e trabalhamos.

O surgimento de agentes inteligentes (de 1995 até a atualidade)

Desde 1995, a inteligência artificial tem experimentado um crescimento significativo na área de agentes inteligentes. Agentes inteligentes são sistemas que tomam decisões e realizam tarefas de forma autônoma com base em informações sensoriais e conhecimento prévio. Eles surgiram como uma extensão da inteligência artificial tradicional, que se concentrava principalmente em tarefas específicas como reconhecimento de fala e visão computacional.

Conforme mencionamos, o Eliza é um agente inteligente criado em 1966 por Joseph Weizenbaum. Considerado um dos primeiros exemplos de inteligência artificial conversacional, ele foi projetado como uma simulação de uma terapeuta psicológica e é um marco histórico na história da inteligência artificial conversacional. Sua influência pode ser vista em muitos dos agentes inteligentes de hoje, incluindo assistentes virtuais como Siri e Alexa. Em 1995, foi portado para a web, tornando-se acessível a um público global e tornando-se ainda mais popular. Até hoje, muitas pessoas experimentam o programa como uma forma de explorar as possibilidades da inteligência artificial conversacional.

Em meados dos anos 2000, os agentes inteligentes começaram a ser aplicados em aplicativos comerciais, como assistentes virtuais e robôs de atendimento ao cliente. O avanço na capacidade de processamento de dados e a disponibilidade de grandes conjuntos de dados permitiram que os agentes inteligentes se tornassem cada vez mais sofisticados.

Desde então, a inteligência artificial tem experimentado um crescimento significativo com a popularidade de agentes inteligentes, como assistentes virtuais, chatbots, carros autônomos e drones. A pesquisa em agentes inteligentes está em constante evolução e está sendo aplicada em uma variedade cada vez maior de campos, como saúde, finanças e segurança cibernética.

Disponibilidade de conjuntos de dados muito grandes (2000 até a atualidade)

A inteligência artificial estuda como fazer com que as máquinas realizem tarefas que normalmente requerem inteligência humana, como aprendizado, raciocínio e compreensão. A disponibilidade de conjuntos de dados muito grandes, conhecidos como big data, desde 2000 tem permitido que os algoritmos de aprendizado de máquina sejam aplicados com sucesso em uma ampla variedade de problemas, desde a previsão de vendas até a detecção de fraudes. O aumento da capacidade de processamento de dados e a diminuição dos custos dos computadores também desempenharam um papel importante na popularidade da IA. Isso tem permitido que as empresas e os cientistas de dados utilizem técnicas de aprendizado automático para extrair insights valiosos de grandes conjuntos de dados.

Podemos aplicar IA em diversas atividades e vários subcampos de pesquisa, como no planejamento autônomo e no escalonamento. O programa Remote Agent da Nasa foi o primeiro programa de planejamento autônomo de bordo a controlar o escalonamento de operações de uma nave espacial (JÓNSSON *et al.*, 2000).

1.3 Pesquisas realizadas

A inteligência artificial é um campo em rápido crescimento que tem o potencial de transformar muitos aspectos de nossa vida cotidiana. As pesquisas mais recentes na IA estão explorando novas formas de tornar as máquinas mais inteligentes e capazes de realizar tarefas antes consideradas exclusivas dos seres humanos.

Uma das áreas mais promissoras da IA é a aprendizagem profunda, também conhecida como aprendizagem de máquina. A aprendizagem profunda é uma técnica de IA que permite que as máquinas aprendam por meio da análise de grandes conjuntos de dados, sem a necessidade de programação explícita. Isso tem permitido que as máquinas atinjam níveis surpreendentes de precisão em tarefas como reconhecimento de imagens e processamento de linguagem natural.

Outra área em rápido crescimento é a inteligência artificial generalizada (AGI, na sigla em inglês). A AGI é a capacidade de uma máquina de realizar qualquer tarefa que um ser humano possa fazer. Enquanto a maioria das aplicações atuais da IA são especializadas em uma tarefa específica, a AGI busca criar máquinas capazes de realizar qualquer tarefa, assim como os seres humanos. Isso inclui tarefas como raciocínio, planejamento, aprendizado, compreensão de linguagem natural e percepção.

Além disso, pesquisadores estão explorando novas formas de tornar as máquinas mais autônomas. Isso inclui a criação de sistemas de aprendizado de reforço, que permitem que as máquinas aprendam a partir de suas próprias experiências, e a criação de sistemas de tomada de decisão autônomos, que possibilitam que as máquinas tomem decisões sem intervenção humana.

Outra área de pesquisa importante é a IA ética e governança, crucial para garantir que a IA seja usada de forma responsável e segura. Isso inclui questões como privacidade, segurança, desigualdade e responsabilidade.

Além disso, a combinação da inteligência artificial com outras tecnologias emergentes, como a internet das coisas (IoT) e a robótica, está criando oportunidades para aplicações inovadoras e transformadoras.

O processamento de linguagem natural é outra área de pesquisa em IA que tem sido amplamente estudada. Isso inclui técnicas para permitir que os computadores entendam e gerem texto humano. Algo útil em aplicações como assistentes virtuais, chatbots e sistemas de tradução automática. Uma das áreas mais recentes de pesquisa no processamento de linguagem natural é a geração automática de texto, em que os computadores são treinados para gerar texto humano legível e coerente.

A visão computacional é outra área de pesquisa em IA que tem ganhado muita atenção recentemente. Essa área se concentra em permitir que os computadores entendam e interpretem imagens e vídeos, o que é útil em aplicações como reconhecimento facial, seguimento de objetos e condução autônoma.

1.4 Softwares inteligentes

Os softwares inteligentes que usam inteligência artificial são programas que foram projetados para realizar tarefas que normalmente requerem inteligência humana. Esses softwares podem ser usados em muitas áreas, incluindo negócios, saúde, educação e entretenimento.

Um exemplo de software inteligente é o assistente virtual, como o Amazon Alexa ou o Google Assistant. Esses programas usam a IA para entender e responder às perguntas dos usuários em linguagem natural, além de realizar tarefas como reproduzir música, controlar dispositivos domésticos inteligentes e fornecer informações sobre o tempo.

Outro exemplo é o software de reconhecimento de imagens, como o usado em sistemas de segurança de câmeras ou em aplicativos de reconhecimento facial. Esses programas usam técnicas de aprendizado profundo para reconhecer rostos e outros objetos em imagens, permitindo que as máquinas tomem decisões baseadas nessas informações.

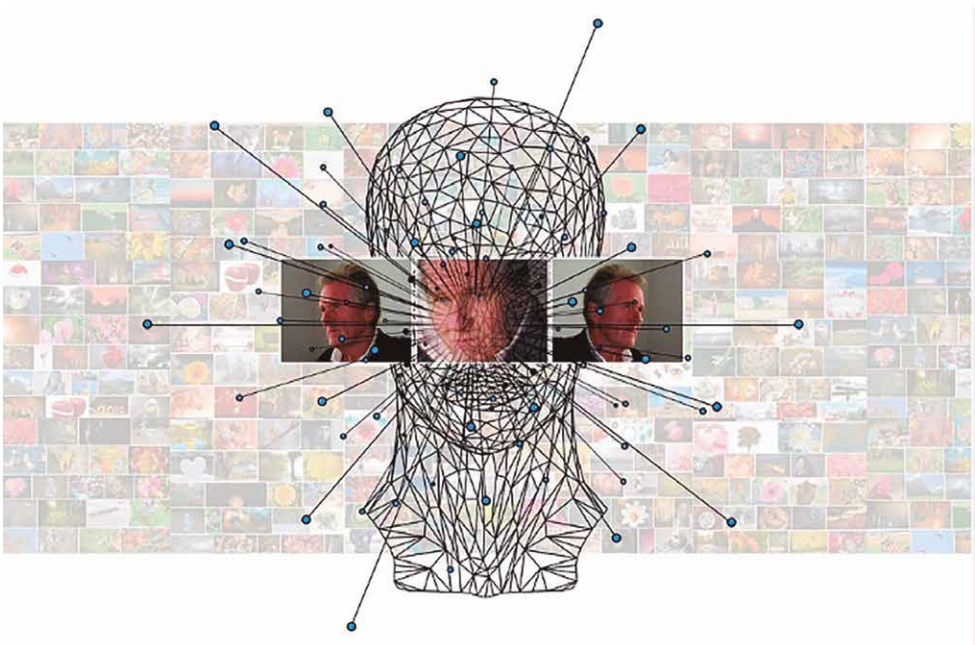


Figura 3

Disponível em: https://cutt.ly/f8DYCMd. Acesso em: 7 mar. 2023.

O chatbot é outro exemplo de software inteligente que usa IA. Ele é projetado para conversar com os usuários em linguagem natural e pode ser usado para atender a perguntas de clientes, ajudar a navegar em um site ou fornecer suporte técnico.

Além disso, a IA está sendo usada em softwares de análise de dados, como os sistemas de business intelligence (BI), que ajudam as empresas a entender e tomar decisões com base em grandes conjuntos de dados.

Business intelligence (BI) é um conjunto de técnicas e ferramentas que visam ajudar as empresas a transformar dados em informações úteis e significativas para a tomada de decisões estratégicas.

O BI envolve a coleta, análise e visualização de dados para permitir que as empresas compreendam melhor seus processos de negócios e identifiquem tendências e padrões que possam ajudá-las a tomar decisões informadas. As informações coletadas podem ser de diversas fontes, como bancos de dados internos da empresa, dados de mercado, dados de concorrentes, dados de clientes, entre outros.

As ferramentas de BI permitem a criação de relatórios, painéis e dashboards personalizados que fornecem aos usuários acesso rápido e fácil a informações importantes. Esses relatórios e dashboards são interativos e permitem que os usuários façam perguntas e obtenham respostas em tempo real.

Alguns exemplos de aplicações de BI incluem análise de vendas, análise de mercado, análise de desempenho de produtos, análise financeira e previsão de tendências de mercado. As empresas também podem usar o BI para identificar áreas que precisam de melhoria e para monitorar o desempenho em relação a metas e objetivos.

O BI é uma ferramenta importante para a tomada de decisões empresariais, pois permite que as empresas tomem decisões baseadas em fatos e dados em vez de apenas em intuição ou suposições. Ele ajuda a melhorar a eficiência e a eficácia dos processos de negócios, fornecendo insights valiosos que podem ser usados para melhorar o desempenho e a rentabilidade da empresa.

Outra área na qual a IA está sendo usada é na automação de processos de negócios (BPA, na sigla em inglês), para automatizar tarefas repetitivas, como a classificação de documentos e a triagem de e-mails.

A inteligência artificial está sendo usada em uma variedade de softwares para realizar tarefas que normalmente requerem inteligência humana. Desde assistentes virtuais até sistemas de análise de dados, os softwares inteligentes estão tornando as máquinas mais capazes e transformando muitos aspectos de nossa vida cotidiana.

2 AGENTES INTELIGENTES

Um agente inteligente é uma entidade capaz de perceber o seu ambiente, tomar decisões e realizar ações com o objetivo de atingir metas específicas. Esses agentes são projetados para funcionar autonomamente e aprender a partir de sua interação com o ambiente, a fim de melhorar sua performance ao longo do tempo.

Um agente percebe o ambiente através de sensores e outras formas de entrada de dados. Esses dados são processados e interpretados pelo agente para produzir uma representação interna do estado atual do ambiente. O agente pode usar essa representação para tomar decisões sobre as ações a serem realizadas a fim de atingir seus objetivos.

2.1 Estrutura dos agentes

Os agentes são entidades que tomam ações no ambiente para alcançar objetivos específicos. A estrutura de um agente geralmente inclui:

- **sensores** que permitem ao agente perceber o ambiente ao seu redor;
- **ações** que permitem ao agente interagir com o ambiente;
- **uma lógica de tomada de decisão** que permite ao agente escolher a ação mais adequada para alcançar seu objetivo;
- **um objetivo ou metas** que definem o que o agente está tentando alcançar.

A figura a seguir apresenta o esquema de um agente e como interagir com ambientes por meio de sensores e atuadores:

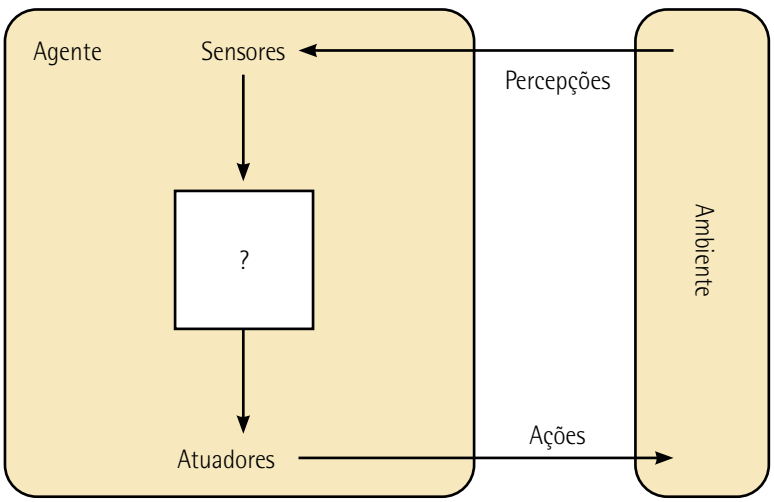


Figura 4 – Esquema de um agente

Fonte: Russell e Norvig (2013, p. 31).

Para exemplificar melhor os agentes podemos citar três exemplos:

Quadro 3

Agente	Sensores	Atuadores
Humano	- Olhos - Ouvidos - Outros órgãos	- Mãos - Pernas - Boca - Outras partes do corpo

Agente	Sensores	Atuadores
Robótico	- Câmeras - Detectores de infravermelho	- Vários motores
Software	- Entrada do teclado - Conteúdo de arquivos - Pacotes vindos da rede	- Tela - Disco - Envio de pacotes pela rede

Além disso, os agentes podem ser classificados como simples ou complexos, de acordo com sua capacidade de percepção, raciocínio e tomada de decisão.

Os agentes simples são aqueles que possuem percepção limitada e capacidade de raciocínio, enquanto os agentes complexos são os que possuem percepção e raciocínio avançados e podem lidar com situações incertas e imprevisíveis.

Medida de desempenho

A medida de desempenho de um agente inteligente é a maneira como avaliamos a qualidade do comportamento ou das ações realizadas pelo agente em relação a um objetivo ou conjunto de objetivos.

Essa medida é usada para avaliar a eficácia de um agente em realizar uma tarefa específica ou alcançar um objetivo particular. Por exemplo, se o objetivo do agente é jogar xadrez, a medida de desempenho pode ser a taxa de vitórias contra um oponente humano ou a pontuação obtida em um conjunto de jogos contra uma variedade de adversários.

Em geral, a medida de desempenho de um agente deve ser claramente definida, ser mensurável e estar relacionada aos objetivos da tarefa. Ela é frequentemente usada com uma função de recompensa, que define o que é "bom" ou "ruim" para o agente, ajudando-o a escolher ações que maximizem a medida de desempenho ao longo do tempo.

A medida de desempenho é uma parte fundamental do processo de aprendizado de máquina, pois ajuda a guiar a otimização do agente através do ajuste de seus parâmetros ou estratégias de tomada de decisão.

Uma medida de desempenho adequada é essencial para que o agente inteligente possa aprender a executar a tarefa da melhor forma possível. Isso ocorre porque o agente busca maximizar a medida de desempenho ao longo do tempo e ajusta seu comportamento para alcançar esse objetivo.

No entanto, é importante ressaltar que a escolha da medida de desempenho pode ter implicações significativas no comportamento do agente. Por exemplo, se a medida de desempenho para um carro autônomo é simplesmente minimizar o tempo de viagem, o agente pode ignorar a segurança e correr riscos desnecessários para alcançar o objetivo. Portanto, a escolha da medida de desempenho deve ser cuidadosa e levar em consideração as implicações éticas e sociais do comportamento do agente.

Algumas medidas de desempenho comuns incluem taxa de acertos, tempo de execução, taxa de falhas, taxa de economia, entre outras. Em alguns casos, pode ser desafiador definir uma medida de desempenho precisa e objetiva, especialmente para tarefas mais complexas que envolvem decisões subjetivas. Nesses casos, pode ser necessário usar uma combinação de medidas de desempenho, bem como incorporar a opinião humana na avaliação do desempenho do agente.

A medida de desempenho ajuda a avaliar a eficácia de um agente em realizar uma tarefa específica. A escolha da medida de desempenho adequada é essencial para que o agente possa aprender e melhorar continuamente seu comportamento.

Especificar as Peas

Peas é um acrônimo que representa um modelo de quatro componentes que descreve a especificação de um agente inteligente. Os quatro componentes são:

- **P (Performance)**: a medida de desempenho usada para avaliar a eficácia do agente em realizar uma tarefa. A medida de desempenho deve estar relacionada aos objetivos da tarefa e pode incluir métricas como tempo de resposta, precisão, economia, segurança, entre outros.

- **E (Environment)**: o ambiente em que o agente opera e interage. O ambiente pode ser físico, virtual ou ambos, e pode incluir vários elementos como objetos, outros agentes, obstáculos, informações, entre outros.

- **A (Actuators)**: os atuadores são os mecanismos que permitem que o agente realize ações no ambiente. Os atuadores podem incluir dispositivos físicos, como braços robóticos, bem como softwares que controlam a interação com o ambiente.

- **S (Sensors)**: os sensores são os mecanismos que permitem que o agente perceba o ambiente e receba informações relevantes para realizar ações. Os sensores podem incluir câmeras, sensores de temperatura, microfones, entre outros.

Juntos, os quatro componentes Peas fornecem uma especificação completa de um agente inteligente e são usados para projetar, desenvolver e avaliar sistemas de inteligência artificial em uma variedade de contextos. A especificação Peas é uma ferramenta útil para entender os requisitos e limitações do agente em relação ao ambiente e à tarefa que ele precisa realizar.

O modelo Peas é usado para ajudar a identificar as características necessárias para um agente realizar uma tarefa específica e, assim, orientar o desenvolvimento de sistemas de inteligência artificial mais eficazes e eficientes. Ele também ajuda a definir a medida de desempenho apropriada para avaliar o sucesso do agente em realizar a tarefa.

A especificação Peas pode ser usada em uma ampla variedade de aplicações de inteligência artificial, desde jogos até sistemas de controle industrial e robótica. Por exemplo, um sistema de monitoramento de temperatura de um reator nuclear pode ser especificado usando o modelo Peas da seguinte maneira:

- **P (Performance)**: a medida de desempenho pode ser a precisão do monitoramento da temperatura do reator nuclear. O objetivo do agente é garantir que a temperatura permaneça dentro de limites seguros.

- **E (Environment)**: o ambiente é o reator nuclear e todos os seus componentes. O ambiente é altamente perigoso e, portanto, é crucial que o agente opere com segurança.

- **A (Actuators)**: os atuadores incluem bomba hidráulica e ventiladores que podem ajustar o nível de refrigeração do reator, se necessário.

- **S (Sensors)**: os sensores incluem termômetros e outros dispositivos de medição de temperatura.

Ao especificar o agente inteligente dessa maneira, podemos projetar um sistema de inteligência artificial que pode monitorar com precisão a temperatura do reator nuclear e tomar medidas preventivas em caso de emergência. O modelo Peas fornece uma estrutura útil para especificar a funcionalidade de um agente e definir a medida de desempenho apropriada para avaliar sua eficácia na realização da tarefa.

2.2 Tipos de agentes

Focamos até o momento no comportamento dos agentes. Vamos focar agora no funcionamento interno desses agentes. O trabalho da IA é desenvolver o programa do agente implementando funções que mapeiam percepções em ação. O programa que desenvolvemos deve estar de acordo com a arquitetura do agente (considerando um dispositivo de computador com sensores e atuadores físicos). Dessa forma, se o programar sugerir a ação de CAMINHAR, a arquitetura deve estar preparada para essa ação.

Agente = Arquitetura + Programa

Descreveremos quatro tipos básicos de programas de agentes que incorporam os princípios subjacentes a quase todos os sistemas inteligentes:

- agentes reativos simples;

- agentes reativos baseados em modelo;

- agentes baseados em objetivos;

- agentes baseados na utilidade.

Agentes reativos simples

Agentes reativos simples são uma subclasse de agentes que tomam decisões baseadas apenas na percepção atual, sem se preocupar com o estado anterior do ambiente. Esses agentes geralmente possuem uma estrutura simples, que inclui sensores, ações e uma lógica de tomada de decisão simples.

Eles não possuem um estado interno, ou seja, não armazenam informações sobre o ambiente anterior e nem têm a capacidade de planejar ações futuras baseadas em objetivos.

Agentes reativos simples são usados em aplicações nas quais a complexidade do ambiente é baixa e as ações do agente são baseadas principalmente na percepção atual. Eles são comumente utilizados em sistemas de robótica simples, como robôs de limpeza, e em jogos simples, como jogos de plataforma.

Por serem simples, esses agentes possuem uma baixa complexidade de tempo e espaço, ou seja, são rápidos e não precisam de muita memória para funcionar.

A figura a seguir apresenta a estrutura de um programa geral em forma esquemática, demonstrando o modo como as regras condição-ação proporcionam ao agente executar a conexão entre percepção e ação. Para denotar o estado interno atual do processo de decisão do agente, utilizamos retângulos e elipses para representar as informações suplementares usadas no processo.

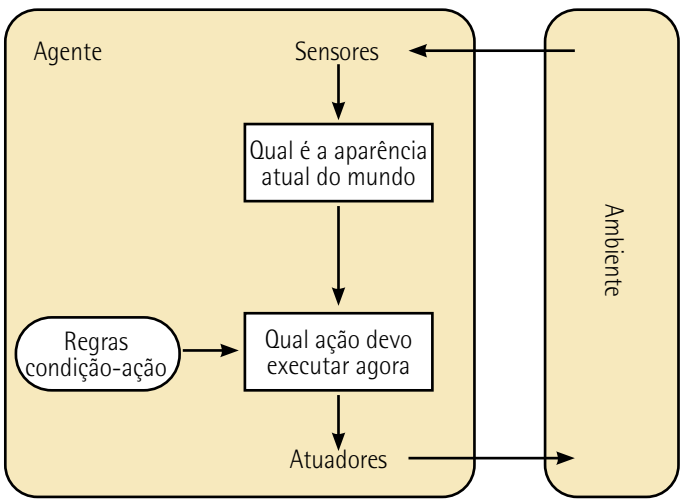


Figura 5 – Diagrama esquemático de um agente reativo simples

Adaptada de: Russell e Norvig (2013, p. 43).

O programa do agente é apresentado na figura a seguir. A função INTERPRETAR-ENTRADA gera uma descrição abstrata do estado atual a partir da percepção. A função REGRA-CORRESPONDENTE retorna à primeira regra no conjunto de regras que corresponde à descrição de estado dada. A descrição em termos de "regras" e "correspondência" é apenas conceitual; as implementações reais podem ser tão simples quanto uma coleção de portas lógicas que compõem um circuito booleano.

```
função AGENTE-REATIVO-SIMPLES (percepção) retorna uma ação
  variáveis estáticas: regras, um conjunto de regras condição-ação
  estado ← INTERPRETAR-ENTRADA (percepção)
  regra ← REGRA-CORRESPONDENTE (estado, regras)
  ação ← AÇÃO-DA-REGRA [regra]
  retornar ação
```

Figura 6 – Agente reativo simples. Ele age de acordo com uma regra cuja condição corresponde ao estado atual definido pela percepção

Fonte: Russell e Norvig (2013, p. 43).

Exemplo: a figura a seguir representa o mundo de um aspirador de pó com apenas duas salas para limpar (Direita e Esquerda):

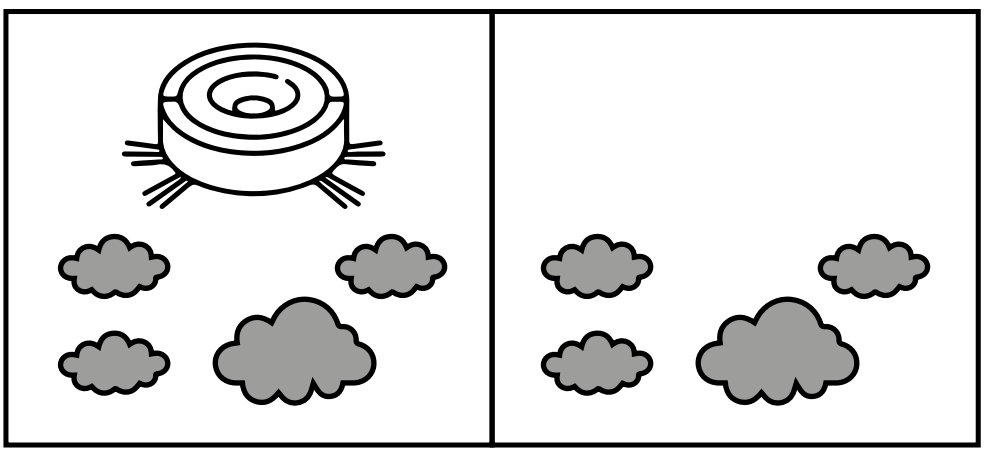


Figura 7 – Mundo de um aspirador de pó com apenas dois locais para limpar

Poderíamos então escrever o programa do agente para um agente reativo simples no ambiente de aspirador de pó de dois estados, conforme a figura a seguir:

```
função AGENTE-ASPIRADOR-DE-PÓ-REATIVO ([sala, situação]) retorna uma ação*
  se situação = Sujo então retorna Aspirar
  senão se sala = Esquerda então retorna Direita
  senão se sala = Direita então retorna Esquerda
```

Figura 8 – Programa do agente para um agente reativo simples no ambiente de aspirador de pó de dois estados

Fonte: Russell e Norvig (2013, p. 43).

Agentes reativos baseados em modelo

Agentes reativos baseados em modelo são uma subclasse de agentes que além de tomar decisões baseadas apenas na percepção atual, também mantêm um modelo interno do ambiente. Eles armazenam informações sobre o estado anterior do ambiente e utilizam essas informações para tomar decisões futuras. Dessa forma, eles são capazes de lidar com situações incertas e imprevisíveis, já que possuem um modelo interno do ambiente.

Os agentes reativos baseados em modelo são usados em aplicações em que a complexidade do ambiente é moderada e as ações do agente são baseadas tanto na percepção atual quanto no estado anterior do ambiente. Eles são comumente utilizados em sistemas de robótica avançados, como robôs autônomos, e em jogos mais complexos, como jogos de estratégia.

A complexidade computacional dos agentes reativos baseados em modelo é geralmente maior do que a dos agentes reativos simples, pois eles precisam manter e atualizar o modelo interno do ambiente.

A figura a seguir apresenta a estrutura do agente reativo baseado em modelo com seu estado interno, demonstrando como é feita a combinação da percepção atual e do estado interno antigo para gerar a descrição atualizada do estado atual, baseado no modelo do agente de como o mundo funciona.

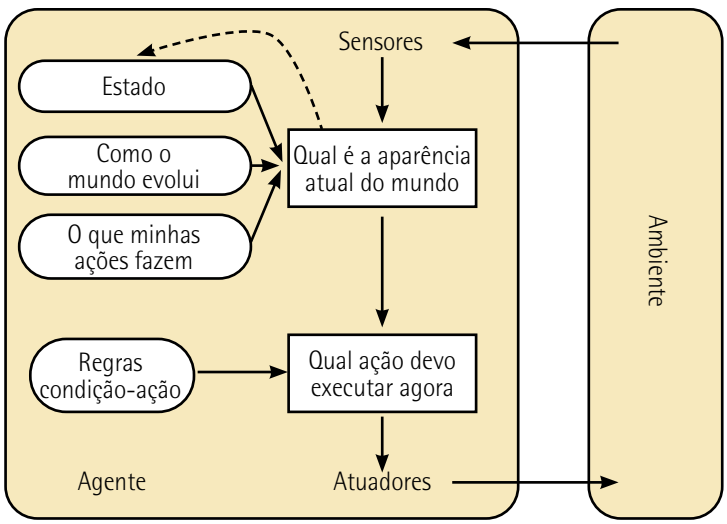


Figura 9 – Agente reativo baseado em modelo

Adaptada de: Russell e Norvig (2013, p. 44).

A figura a seguir apresenta um agente reativo baseado em modelo. Ele mantém o estado atual do mundo usando um modelo interno. Em seguida, ele escolhe uma ação da mesma maneira que o agente reativo simples. A função ATUALIZAR-ESTADO é encarregada pela criação da descrição do novo estado interno.

```
função AGENTE-REATIVO-BASEADO-EM-MODELOS (percepção) retorna uma ação
  persistente: estado, a concepção do agente do estado atual do mundo
               modelo, uma descrição de como o próximo estado depende do estado atual e da ação
               regras, um conjunto de regras condição-ação
               ação, a ação mais recente, inicialmente nenhuma
  estado ← ATUALIZAR-ESTADO (estado, ação, percepção, modelo)
  regra ← REGRA-CORRESPONDENTE (estado, regras)
  ação ← regra, AÇÃO
  retornar ação
```

Figura 10 – Agente reativo baseado em modelo

Fonte: Russell e Norvig (2013, p. 44).

Agentes baseados em objetivos

Agentes baseados em objetivos são uma subclasse de agentes que possuem metas ou objetivos específicos a serem alcançados. Esses agentes possuem uma estrutura mais complexa do que os agentes reativos simples ou baseados em modelo, incluindo sensores, ações, uma lógica de tomada de decisão avançada e uma capacidade de planejar ações futuras baseadas em objetivos.

Os agentes baseados em objetivos são usados em aplicações em que a complexidade do ambiente é alta e as ações do agente são baseadas tanto na percepção atual quanto no estado anterior do ambiente, além de objetivos futuros a serem alcançados. Eles são comumente utilizados em aplicações avançadas como inteligência artificial para jogos, robótica autônoma, assistentes virtuais, entre outros.

A complexidade computacional dos agentes baseados em objetivos é geralmente maior do que a dos agentes reativos simples ou baseados em modelo, pois eles precisam planejar ações futuras baseadas em objetivos e lidar com situações incertas e imprevisíveis.

A figura a seguir apresenta a estrutura de um agente baseado em modelos e orientado pelos objetos. O estado do mundo é monitorado, assim como um conjunto de objetivos que devem ser atingidos, e seleciona uma ação que levará à realização de seus objetivos.

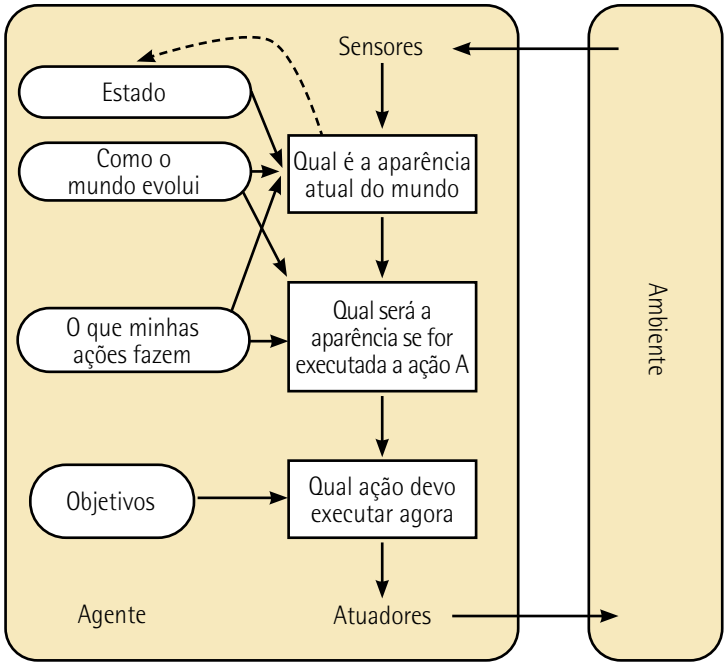


Figura 11 – Agente baseado em modelos e orientado pelos objetivos

Adaptada de: Russell e Norvig (2013, p. 45).

Agentes baseados em utilidade

Agentes baseados em utilidade são uma subclasse de agentes baseados em objetivos que utilizam a Teoria da Utilidade para tomar decisões. Essa teoria se baseia na ideia de que a escolha de uma ação deve ser baseada na utilidade esperada dessa ação, ou seja, no valor esperado de alcançar o objetivo.

Para calcular a utilidade de uma ação, os agentes baseados em utilidade usam uma função de utilidade, que atribui um valor numérico a cada estado possível do ambiente. Essa função é geralmente definida pelo designer do sistema ou aprendida a partir de dados de treinamento.

Os agentes baseados em utilidade são usados em aplicações em que a complexidade do ambiente é alta e as ações do agente são baseadas tanto na percepção atual quanto no estado anterior do ambiente, além de objetivos futuros a serem alcançados e a utilidade dessas ações.

A complexidade computacional dos agentes baseados em utilidade é geralmente maior do que a dos agentes baseados em objetivos, pois eles precisam calcular a utilidade esperada das ações e lidar com situações incertas e imprevisíveis.

A figura a seguir apresenta a estrutura de um agente baseado em modelo e orientado para a utilidade. Nele, é utilizado um modelo do mundo com uma função utilidade que avalia suas prioridades entre estados do mundo. O próximo passo é escolher a ação que leva à melhor utilidade desejada, na qual a

utilidade desejada é calculada pela média entre todos os estados resultantes possíveis, ponderados pela probabilidade do resultado.

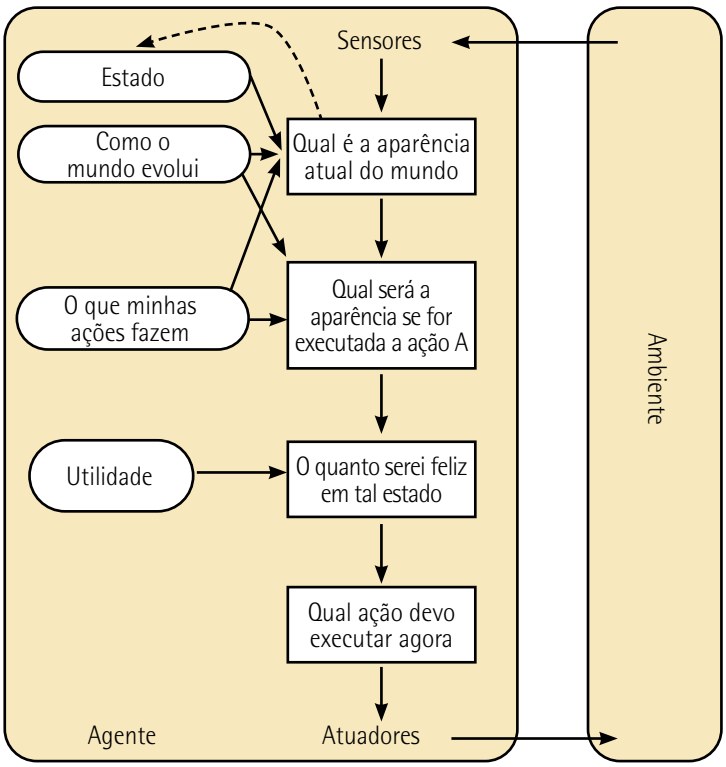


Figura 12 – Agente baseado em modelo e orientado para a utilidade

Adaptada de: Russell e Norvig (2013, p. 46).

Agentes com aprendizagem

Agentes com aprendizagem são uma subclasse de agentes que são capazes de aprender e melhorar suas ações ao longo do tempo. Eles usam algoritmos de aprendizado de máquina para ajustar suas ações baseadas em feedback de seu desempenho. Dessa forma, eles podem adaptar-se a mudanças no ambiente e melhorar sua capacidade de alcançar seus objetivos.

Os agentes com aprendizagem podem ser classificados como supervisionados, não supervisionados ou de reforço:

- **Agentes supervisionados**: são aqueles que aprendem a partir de dados rotulados, em que o objetivo é aprender uma função que mapeia entradas para saídas.
- **Agentes não supervisionados**: são aqueles que aprendem a partir de dados não rotulados, em que o objetivo é descobrir padrões ocultos nos dados.
- **Agentes de reforço**: são aqueles que aprendem através da tentativa e erro, recebendo recompensas ou punições para cada ação tomada.

Os agentes com aprendizagem são usados em aplicações em que a complexidade do ambiente é alta e as ações do agente são baseadas tanto na percepção atual quanto no estado anterior do ambiente, além de objetivos futuros a serem alcançados e a utilidade dessas ações, mas precisam aprender ao longo do tempo e adaptar-se a mudanças no ambiente. Eles são comumente utilizados em sistemas de inteligência artificial para jogos, robótica autônoma, inteligência artificial para finanças, inteligência artificial para saúde, entre outros.

A figura a seguir apresenta a estrutura de um modelo geral de agentes com aprendizagem. Podemos observar quatro componentes conceituais:

- **elemento de aprendizado**: responsável pela execução de aperfeiçoamentos;
- **elemento de desempenho**: responsável pela seleção de ações externas;
- **crítico**: informa ao elemento de aprendizado como o agente está se comportando em relação a um padrão fixo de desempenho;
- **gerador de problemas**: responsável por sugerir ações que levarão a experiências novas e informativas.

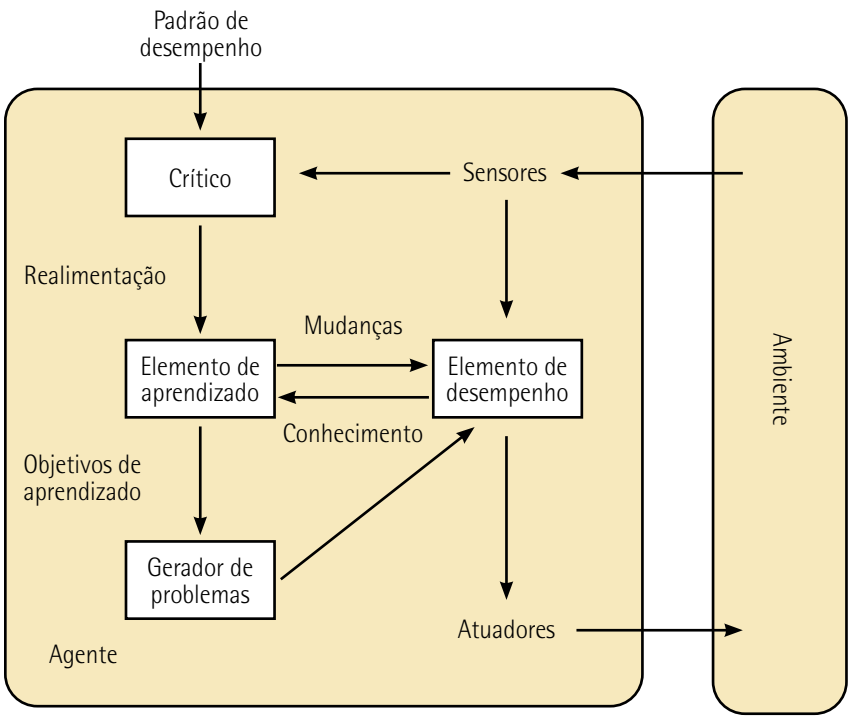


Figura 13 – Modelo geral de agentes com aprendizagem

Fonte: Russell e Norvig (2013, p. 47).

2.3 Sistemas multiagente

Sistemas multiagente são sistemas compostos de múltiplos agentes que interagem entre si e com o ambiente. Esses agentes podem ser individuais ou coletivos, e cada um pode ter sua própria estrutura, lógica de tomada de decisão e objetivos.

São usados em aplicações em que a complexidade do ambiente é muito alta e as ações de um único agente são insuficientes para alcançar os objetivos desejados. Ao dividir a tarefa entre vários agentes, é possível resolver problemas mais complexos e lidar com incertezas e imprevisibilidade.

Também são amplamente utilizados em várias áreas, como inteligência artificial para jogos, robótica autônoma, inteligência artificial para transporte, inteligência artificial para finanças, inteligência artificial para saúde, entre outros, assim como em simulações, como simulações de tráfego, simulações de sistemas econômicos, simulações de ecossistemas, entre outros.

A complexidade computacional dos sistemas multiagente é geralmente maior do que a dos agentes individuais, pois eles precisam lidar com a interação entre os agentes e a coordenação de suas ações.

Os sistemas multiagente podem ser softwares, robôs ou até mesmo humanos, e trabalham juntos para atingir objetivos comuns ou para resolver problemas complexos. Eles são projetados para funcionar de forma descentralizada, sem um único ponto de controle central, e para se adaptarem dinamicamente ao ambiente em que operam. Esses sistemas são comumente usados em aplicações como robótica, inteligência ambiental, jogos e simulações.

Inteligência artificial distribuída (IAD)

Inteligência artificial distribuída é uma abordagem para a inteligência artificial que consiste em dividir tarefas complexas entre múltiplos computadores ou dispositivos, que trabalham juntos de forma coordenada para alcançar um objetivo comum. Essa abordagem permite que sistemas de inteligência artificial possam lidar com grandes volumes de dados e processamento intensivo, além de permitir escalabilidade e tolerância a falhas.

Veremos na sequência que os sistemas de inteligência artificial distribuídos podem ser classificados em dois tipos: sistemas de inteligência artificial distribuídos centralizados e sistemas de inteligência artificial distribuídos descentralizados.

- **sistemas de inteligência artificial distribuídos centralizados** possuem um único ponto central de controle que gerencia a distribuição de tarefas e a comunicação entre os nós;

- **sistemas de inteligência artificial distribuídos descentralizados** não possuem um único ponto central de controle, e cada nó age de forma autônoma.

Comunicação e coordenação em sistemas multiagente

Comunicação e coordenação em sistemas multiagente são fundamentais para o funcionamento eficaz desses sistemas. A comunicação permite que os agentes troquem informações e coordenem suas ações para atingir objetivos comuns. A coordenação, por sua vez, é o processo de garantir que as ações dos agentes sejam compatíveis e complementares, evitando conflitos e maximizando a eficiência do sistema como um todo. A comunicação e a coordenação são geralmente alcançadas por meio de protocolos de comunicação, acordos de coordenação e mecanismos de tomada de decisão.

Saiba mais

Leia o artigo a seguir. Nele, o autor discute como coordenar o problema da disciplina de inteligência artificial distribuída (IAD). Ele defende que para fazer avanços é importante que as teorias e princípios que orientam essa atividade central sejam analisados de forma sistemática e rigorosa. Para tanto, ele modela comunidades de agentes usando um formalismo de busca de metas distribuídas.

O'HARE, G. M. P.; JENNINGS, N. R. *Foundations of distributed artificial intelligence*. Nova Jersey: Wiley, 1996.

3 RESOLUÇÃO DE PROBLEMAS UTILIZANDO ALGORITMOS DE BUSCA

A resolução de problemas e os algoritmos de busca são duas áreas centrais da ciência da computação. A resolução de problemas é o processo de encontrar uma solução para um problema usando técnicas de análise e raciocínio. Os algoritmos de busca são um conjunto de técnicas usadas para encontrar soluções para problemas em que é necessário procurar entre muitas opções.

Os algoritmos de busca são utilizados em muitas áreas da ciência da computação, como em jogos, sistemas de recomendação, sistemas de planejamento e robótica. Eles são usados para encontrar soluções para problemas complexos em que é necessário procurar em uma grande quantidade de dados para encontrar a melhor resposta possível. Esses algoritmos podem ser divididos em duas categorias principais: busca não informada e busca informada.

A busca não informada é um tipo de algoritmo de busca que não tem dados adicionais sobre o problema, além dos brutos. Esses algoritmos geralmente usam técnicas como busca em largura e busca em profundidade para explorar todas as possíveis soluções até encontrar a resposta correta. A busca em largura examina todas as opções em um nível antes de passar para o próximo nível, enquanto a busca em profundidade examina todas as opções em um ramo antes de passar para o próximo ramo.

Já a busca informada, também conhecida como busca heurística, usa informações adicionais sobre o problema para encontrar a solução mais rapidamente. Essas informações podem ser usadas para orientar a busca na direção certa, eliminando opções que não levam a uma solução válida.

Lembrete

Heurística é um processo de tomada de decisão baseado em regras simplificadas ou intuições. É um método para resolver problemas que busca uma solução rápida e eficiente, mesmo que não seja necessariamente a solução ideal ou ótima.

Um exemplo de aplicação de algoritmos de busca é a resolução de um quebra-cabeça. Suponha que você precise encontrar uma solução para um cubo mágico. A busca não informada exploraria todas as possíveis soluções até encontrar a resposta correta, o que pode levar muito tempo. Já a busca informada usaria informações adicionais sobre a posição das peças para encontrar a solução mais rapidamente.

Além dos algoritmos de busca não informada e informada, existem outros tipos de algoritmos que podem ser usados para resolver problemas, como algoritmos genéticos, algoritmos de otimização e algoritmos de aprendizado de máquina.

Os algoritmos genéticos são usados para resolver problemas de otimização em que é necessário encontrar a melhor solução dentro de um conjunto de possíveis soluções. Esses algoritmos são baseados na seleção natural e na Teoria da Evolução, em que soluções melhores são selecionadas e combinadas para gerar soluções ainda melhores.

Os algoritmos de otimização são usados para encontrar a melhor solução em um conjunto de possíveis soluções. Eles geralmente usam técnicas matemáticas para encontrar a solução mais próxima possível da solução ideal.

Os algoritmos de aprendizado de máquina são usados para treinar um modelo com dados de entrada e saída conhecidos, de modo que ele possa prever a saída para novos dados de entrada. Eles são usados em áreas como reconhecimento de voz, detecção de spam e análise de sentimentos.

Independentemente do tipo de algoritmo usado, a resolução de problemas é um processo iterativo que envolve a formulação do problema, a seleção do algoritmo mais adequado e a análise da solução encontrada. A escolha do algoritmo certo pode fazer a diferença entre encontrar uma solução rápida e eficiente e ficar preso em uma solução inadequada ou ineficiente. Portanto, é importante conhecer as diferentes técnicas disponíveis e escolher aquela que melhor se adapta ao problema em questão.

Além disso, é importante considerar a complexidade do problema e a quantidade de dados envolvidos. Problemas com grande volume de dados podem exigir algoritmos mais eficientes e escaláveis, enquanto problemas com menos dados podem permitir a utilização de algoritmos mais simples.

Outro fator importante a ser considerado é a qualidade dos dados. Dados inconsistentes ou incompletos podem afetar a eficácia do algoritmo e levar a soluções imprecisas. Por isso, é fundamental garantir que os dados sejam limpos e preparados corretamente antes de aplicar o algoritmo.

Também é importante avaliar a qualidade da solução encontrada pelo algoritmo. Em muitos casos, a solução encontrada pode não ser a ideal, mas ainda assim ser suficientemente boa para resolver o problema. Nesses casos, é essencial avaliar a precisão e a confiabilidade da solução encontrada.

Por fim, é importante lembrar que a resolução de problemas e os algoritmos de busca são áreas em constante evolução, com novas técnicas e algoritmos sendo desenvolvidos o tempo todo. Por isso, é essencial estar sempre atualizado e buscar conhecimento sobre as últimas tendências e técnicas disponíveis para resolver problemas de forma mais eficiente e eficaz.

3.1 Resolução de problemas

Problemas padronizados

Os problemas padronizados são problemas comuns que são amplamente utilizados para avaliar o desempenho de sistemas de inteligência artificial. Eles geralmente incluem tarefas específicas, como reconhecimento de imagem, processamento de linguagem natural e jogos de tabuleiro, e são projetados para testar as habilidades de uma IA em áreas específicas. Os resultados desses problemas são comparáveis entre distintos sistemas, permitindo que os pesquisadores comparem o desempenho de diferentes abordagens e algoritmos.

Russell e Norvig (2013, p. 61) analisam o mundo do aspirador de pó, um exemplo de problema padronizado, e formulam o problema da seguinte forma:

- Estados: o estado é determinado tanto pela posição do agente como da sujeira. O agente está em uma entre duas posições, cada uma das quais pode conter sujeira ou não. Desse modo, há 2 x 22 = 8 estados do mundo possíveis. Um ambiente mais amplo com n posições tem n x 2n estados.
- Estado inicial: qualquer estado pode ser designado como o estado inicial.
- Ações: nesse ambiente simples, cada estado tem apenas três ações: Esquerda, Direita e Aspirar. Ambientes mais amplos podem também incluir Em Cima e Embaixo.
- Modelo de transição: as ações têm seus efeitos esperados, a não ser as ações: mover para a Esquerda no quadrado mais à esquerda, mover para a Direita, no quadrado mais à direita, e aspirar, no quadrado limpo, que não tem nenhum efeito.
- Teste de objetivo: verifica se todos os quadrados estão limpos.
- Custo de caminho: cada passo custa 1 e, assim, o custo do caminho é o número de passos do caminho.

Na figura a seguir podemos observar o espaço de estados completo. Se comparado o mundo do aspirador de pó com o mundo real, esse problema de mundo padronizado tem posições discretas, sujeira discreta, limpeza confiável e nunca se torna sujo depois de limpo.

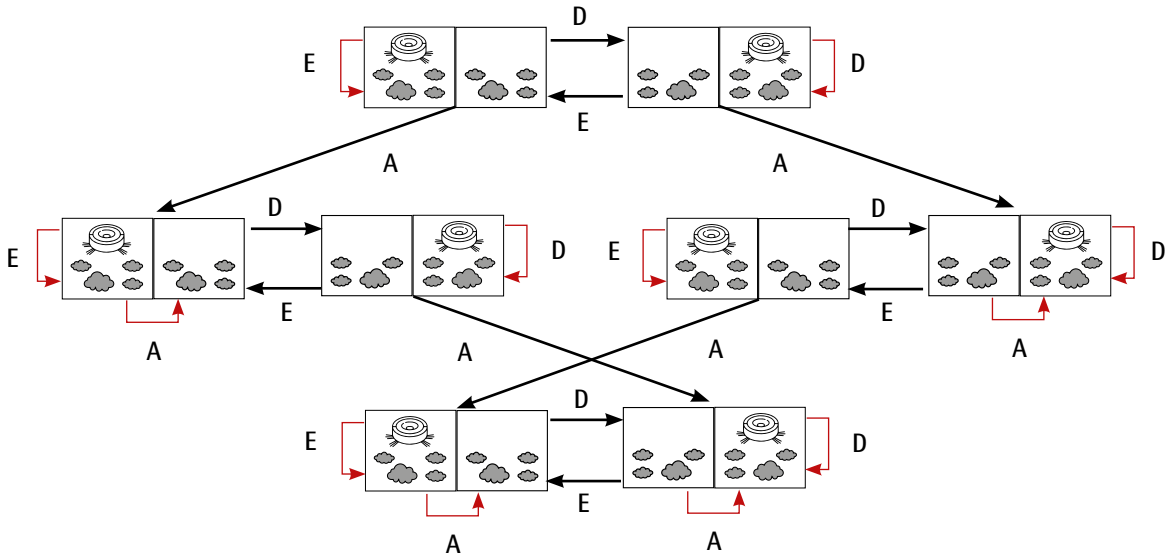


Figura 14 – O espaço de estados para o mundo do aspirador de pó.
Os arcos denotam ações: E = Esquerda, D = Direita, A = Aspirar

Problemas do mundo real

Os problemas do mundo real são problemas que encontramos em nosso ambiente cotidiano e que geralmente são mais complexos e desafiadores do que os problemas padronizados. Eles podem incluir tarefas como conduzir um carro, realizar o reconhecimento de fala em ambientes ruidosos, planejar rotas em um mapa ou tomar decisões em situações incertas. Esses problemas geralmente requerem uma combinação de habilidades, como percepção, raciocínio e aprendizado, para serem resolvidos e são mais difíceis de serem modelados e testados do que os problemas padronizados.

Como exemplo, podemos citar o problema do caixeiro viajante, que é um problema clássico de otimização combinatória em que se busca encontrar o menor caminho possível que um caixeiro viajante pode percorrer para visitar todas as cidades em um conjunto dado, retornando à cidade de origem. O problema é um dos mais conhecidos da Teoria dos Grafos e tem aplicações em diversas áreas, como logística, transporte, planejamento urbano, roteirização, dentre outras.

O desafio do problema é encontrar a rota mais curta possível, considerando todas as cidades do conjunto e retornando à cidade de origem, passando por cada cidade uma única vez. Em outras palavras, o objetivo é minimizar a distância total percorrida pelo caixeiro viajante.

O problema do caixeiro viajante é conhecido como um problema NP-difícil, o que significa que não se conhece um algoritmo eficiente para resolvê-lo em tempo polinomial para instâncias grandes

do problema. Por isso, é comum utilizar heurísticas e algoritmos aproximados para encontrar soluções subótimas em um tempo razoável. Há diversos algoritmos heurísticos conhecidos para o problema, como o algoritmo do vizinho mais próximo, algoritmos genéticos, algoritmos de busca local, dentre outros. É um problema de grande importância prática e teórica. Ele tem sido estudado desde a década de 1930 e possui diversas variações, por exemplo, a inclusão de restrições adicionais, como limitações de tempo ou de recursos, ou ainda a presença de múltiplos caixeiros viajantes.

Embora o problema do caixeiro viajante possa parecer simples para poucas cidades, o número de possibilidades de rotas a serem exploradas cresce exponencialmente com o aumento do número de cidades. Por exemplo, para um conjunto de 10 cidades, existem mais de 3 milhões de rotas possíveis a serem exploradas.

A complexidade do problema torna sua resolução exata inviável para instâncias grandes, pois o tempo necessário para encontrar a solução ótima cresce exponencialmente com o tamanho do conjunto de cidades. Por essa razão, são utilizados algoritmos heurísticos e aproximados, que podem encontrar soluções subótimas em um tempo razoável.

Uma das heurísticas mais simples e eficientes para o problema do caixeiro viajante é o algoritmo do vizinho mais próximo. Neste algoritmo, o caixeiro viajante escolhe sempre a cidade mais próxima disponível para visitar. Embora não garanta a solução ótima, essa heurística muitas vezes produz soluções boas o suficiente para muitas aplicações práticas.

Outra abordagem comum para resolver o problema do caixeiro viajante é o uso de algoritmos genéticos, que são inspirados no processo de evolução biológica. Nesse tipo de algoritmo, a população de soluções é tratada como uma "população" de indivíduos, que são "selecionados" e "reproduzidos" de forma a gerar novas soluções a cada geração. O algoritmo genético pode ser utilizado para encontrar soluções melhores do que o algoritmo do vizinho mais próximo, mas requer mais recursos computacionais.

O problema do caixeiro viajante é um problema fundamental em Teoria dos Grafos e tem aplicações em diversos campos, desde a logística e transporte até a biologia e química. É um exemplo clássico de problema de otimização combinatória, que desafia os limites da computação e requer soluções inteligentes para serem resolvidos de forma eficiente.

Espaço de estados

O espaço de estados é um conceito fundamental na IA que se refere ao conjunto de estados possíveis e suas transições para um determinado problema. Ele é usado para modelar problemas de busca e planejamento, como jogos, robótica e inteligência artificial para jogos.

O espaço de estados é composto de um conjunto de estados, cada um representando uma configuração possível do problema. Esses estados são conectados por ações ou transições, que representam as ações que podem ser tomadas para mudar de um estado para outro. As ações são regidas por regras e restrições, que determinam quais ações são possíveis em cada estado.

O espaço de estados é geralmente representado como um grafo, em que cada nó representa um estado e as arestas representam as transições entre os estados. Isso permite que os algoritmos de busca, como busca em largura e busca em profundidade, sejam aplicados para encontrar soluções para o problema.

O espaço de estados também é usado para planejamento, que é o processo de encontrar uma sequência de ações que leve de um estado inicial para um estado final desejado. Os algoritmos de planejamento, como o algoritmo de busca A*, usam informações sobre o espaço de estados, como a heurística, para encontrar a melhor sequência de ações.

Representação

A representação do espaço de estados é uma etapa fundamental em muitos algoritmos de busca, como a busca em árvore e a busca em grafo. A ideia é representar o problema em questão em termos de estados, em que cada estado representa uma configuração ou situação possível do problema. A partir dessa representação, os algoritmos de busca podem explorar o espaço de estados em busca de soluções para o problema.

A representação do espaço de estados pode ser feita de diferentes maneiras, dependendo do tipo de problema em questão. Algumas das formas mais comuns de representação são:

- **Vetor ou matriz de estado**: para problemas em que o espaço de estados pode ser representado por um conjunto de valores ou variáveis, como no problema do caixeiro viajante, é comum representar o espaço de estados como um vetor ou matriz de valores. Cada posição no vetor ou matriz representa um aspecto ou característica do estado.

- **Grafo de estados**: para problemas em que o espaço de estados é mais complexo ou possui uma estrutura não linear, pode ser útil representar o espaço de estados como um grafo de estados. Nesse tipo de representação, cada nó do grafo representa um estado possível do problema, e as arestas do grafo representam as transições possíveis entre os estados.

- **Conjunto de regras ou axiomas**: para problemas em que o espaço de estados é definido por um conjunto de regras ou axiomas, pode ser útil representar o espaço de estados como um conjunto de regras ou axiomas, que descrevem as transições possíveis entre os estados.

- **Autômatos e máquinas de estado**: para problemas em que o espaço de estados pode ser representado por uma máquina de estados ou um autômato, pode ser útil representar o espaço de estados por meio dessas estruturas. Em um autômato ou máquina de estado, os estados representam as possíveis configurações do problema e as transições entre os estados são representadas por transições de um estado para outro.

- **Árvores de decisão**: para problemas em que o espaço de estados pode ser representado por uma árvore de decisão, pode ser útil representar o espaço de estados por meio dessa estrutura. Cada nó da árvore de decisão representa uma escolha ou decisão que deve ser tomada no processo de resolução do problema, e cada ramo da árvore representa uma sequência de escolhas que levam a uma solução.

Em geral, a escolha da forma de representação do espaço de estados depende do tipo de problema em questão e das características da solução desejada. Alguns problemas podem ter várias formas de representação possíveis, e a escolha da melhor forma de representação pode ser influenciada por questões de desempenho computacional, facilidade de implementação ou outros fatores. Em todo caso, a representação adequada do espaço de estados é um passo fundamental para o sucesso de algoritmos de busca e resolução de problemas.

Independentemente da forma de representação escolhida, é importante que a representação do espaço de estados seja clara e completa, de forma a incluir todas as informações necessárias para que o algoritmo de busca possa explorar o espaço de estados de forma adequada. Além disso, a representação deve ser eficiente e escalável, para que possa ser aplicada em problemas grandes ou complexos.

Desempenho da resolução de problemas

Análise da complexidade é uma técnica utilizada para medir o desempenho de algoritmos em IA. Ela permite avaliar o tempo e o espaço necessário para que um algoritmo execute e forneça uma medida da eficiência do algoritmo. Isso é importante para garantir que os algoritmos possam ser executados de forma rápida e eficiente, especialmente quando lidamos com grandes conjuntos de dados.

Existem várias formas de medir a complexidade de um algoritmo, mas a mais comum é a notação O. A notação O fornece uma estimativa do número de operações que um algoritmo realiza em relação ao tamanho do conjunto de dados de entrada. Por exemplo, um algoritmo com complexidade O(n) realiza uma quantidade de operações proporcional ao tamanho do conjunto de dados. Um algoritmo com complexidade O(1) realiza uma quantidade constante de operações independentemente do tamanho do conjunto de dados.

Outra medida de complexidade comum é a notação Θ (theta), que fornece uma estimativa do limite superior e inferior do número de operações realizadas pelo algoritmo.

É importante notar que a complexidade de um algoritmo pode ser afetada por vários fatores, incluindo a estrutura de dados utilizada, a implementação do algoritmo e até mesmo a linguagem de programação usada. Por isso, é importante analisar a complexidade de um algoritmo em combinação com outras métricas de desempenho, como precisão e velocidade.

Além disso, existe a análise de complexidade de pior caso, melhor caso e caso médio, que fornecem uma avaliação do desempenho do algoritmo em situações específicas. Por exemplo, a complexidade de pior caso é a pior situação possível em que o algoritmo pode ser aplicado, enquanto a complexidade de melhor caso é a melhor situação possível.

A análise de complexidade é uma técnica importante na inteligência artificial para avaliar o desempenho dos algoritmos e garantir que eles possam ser executados de forma rápida e eficiente. É importante levar em conta vários fatores, como estrutura de dados, implementação e linguagem de programação, e combiná-los com outras métricas de desempenho para obter bons resultados.

- **Completude**: a medida de completude é uma métrica usada para avaliar a capacidade de um sistema de inferência lógica de produzir uma resposta para todas as entradas válidas. Em outras palavras, a medida de completude avalia se o sistema é capaz de responder a todas as perguntas válidas dentro de um determinado conjunto de entradas. Essa métrica é frequentemente usada em sistemas de inferência baseados em regras, como sistemas de inferência baseados em lógica de primeira ordem (veremos isso adiante neste livro-texto).

- **Otimização de custo**: a medida de otimização de custo é uma métrica usada para avaliar a eficiência de um algoritmo ou sistema em relação ao uso de recursos. Isso pode incluir recursos como tempo de processamento, memória, energia e outros recursos. A medida de otimização de custo é usada para comparar diferentes algoritmos ou sistemas e determinar qual é o mais eficiente em termos de uso de recursos. É importante notar que, em alguns casos, otimizar o custo pode comprometer a precisão ou a qualidade do resultado.

- **Complexidade de tempo**: a medida de complexidade de tempo é uma métrica usada para avaliar a eficiência de um algoritmo ou sistema em relação ao tempo necessário para executá-lo. A complexidade de tempo é geralmente expressa em termos de uma função matemática que descreve o crescimento do tempo de execução em relação ao tamanho da entrada. Alguns exemplos de notação de complexidade de tempo incluem O(1), O(n), O(n^2) e O(log n), onde n é o tamanho da entrada.

Em geral, quanto menor a complexidade de tempo, melhor é a eficiência do algoritmo ou sistema. Por exemplo, um algoritmo com complexidade de tempo O(1) é considerado muito mais eficiente do que um algoritmo com complexidade de tempo O(n^2), pois o primeiro algoritmo tem um tempo de execução constante, independentemente do tamanho da entrada, enquanto o segundo algoritmo tem um tempo de execução que cresce quadraticamente com o tamanho da entrada.

- **Complexidade de espaço**: a medida de complexidade de espaço se refere à quantidade de memória necessária para armazenar os dados e estruturas de dados utilizadas por um algoritmo. Isso é importante porque a memória é um recurso finito e, portanto, quanto menos memória um algoritmo precisar, melhor será sua eficiência. A complexidade de espaço é medida em unidades como bytes ou bits e geralmente é expressa como uma função do tamanho do problema.

Algoritmos de busca

Os algoritmos de busca são uma importante ferramenta na inteligência artificial que são usados para encontrar soluções para problemas complexos. Esses algoritmos usam uma variedade de técnicas para buscar através de um espaço de busca, que pode ser tudo, desde um simples conjunto de números até um labirinto virtual.

O algoritmo de busca em largura é uma técnica que busca através de todos os nós de um grafo ou árvore de forma sistemática e uniforme. Ele começa pelo nó inicial e adiciona todos os nós adjacentes à lista de nós a serem visitados. Ele então continua visitando cada nó na lista até encontrar a solução desejada ou determinar que não há solução. Esse algoritmo é simples e garante que todos os nós sejam visitados, mas pode ser ineficiente em espaços de busca maiores devido ao grande número de nós que precisam ser visitados.

O algoritmo de busca em profundidade é outra técnica comum, que busca através dos nós de um grafo ou árvore seguindo uma pré-ordem recursivamente, explorando cada ramo completamente antes de passar para o próximo. Ele também começa no nó inicial e adiciona todos os nós adjacentes à pilha. Ele então continua visitando cada nó na pilha até encontrar a solução desejada ou determinar que não há solução. Esse algoritmo pode ser mais eficiente em espaços de busca maiores, mas pode não garantir que todos os nós sejam visitados e pode ser propenso a cair em looping infinito.

Observação

Uma pilha é uma estrutura de dados em que os elementos são adicionados e removidos somente na extremidade oposta. Essa estrutura de dados segue a lógica "último a entrar, primeiro a sair", ou seja, o último elemento adicionado à pilha é o primeiro a ser removido.

As operações básicas em uma pilha são a inserção (também chamada de empilhamento ou push) de um elemento no topo da pilha e a remoção (também chamada de desempilhamento ou pop) do elemento no topo da pilha. Outra operação comum em pilhas é a verificação do elemento no topo da pilha sem removê-lo (também chamada de top).

As pilhas podem ser implementadas em diversas linguagens de programação, usando estruturas de dados como arrays ou listas encadeadas. A escolha da implementação depende das necessidades específicas do algoritmo em questão.

3.2 Busca cega

O algoritmo de busca cega (também chamado de busca sem informação) é um método comum utilizado na inteligência artificial para encontrar soluções para problemas de busca. Esses algoritmos são baseados em visitar todos os estados possíveis de um problema, geralmente usando uma estratégia de busca exaustiva.

A busca em largura é um algoritmo de busca cega que expande os nós de um grafo de busca em largura (como uma árvore) a partir de um nó inicial e visita todos os nós vizinhos antes de se mover para o próximo nível. Isso garante que todos os estados a uma distância específica do nó inicial sejam visitados antes de se mover para estados mais distantes. Isso é útil para problemas em que é desejável encontrar a solução com o menor número de passos.

A busca em profundidade é outro algoritmo de busca cega que expande os nós a partir do nó inicial e visita cada nó o mais profundamente possível antes de voltar e explorar outros caminhos. Isso garante que todos os estados que são "filhos" de um estado específico sejam visitados antes de se mover para outros estados. Isso é útil para problemas em que é desejável encontrar a solução com o menor número de nós.

Em geral, esses algoritmos de busca cega são simples de implementar e são eficientes para problemas com um número limitado de estados possíveis, mas eles podem ser ineficientes para problemas com muitos estados possíveis, pois visitam muitos estados desnecessários.

3.3 Busca em largura

A busca em largura é geralmente usada em problemas de busca em que existe um estado inicial e um estado final, e o objetivo é encontrar o caminho mais curto entre os dois estados. Ela é útil quando não há uma heurística para avaliar a qualidade de um estado, ou quando não é possível prever o estado final.

A busca em largura é implementada usando uma fila para armazenar os nós visitados. O algoritmo começa explorando todos os nós adjacentes ao estado inicial e adicionando-os à fila. Em seguida, o algoritmo remove o primeiro nó da fila e explora seus nós adjacentes, adicionando-os à fila. Isso é repetido até que o estado final seja encontrado ou até que todos os nós tenham sido explorados.

Observação

Uma fila é uma estrutura de dados em que os elementos são adicionados ao final e removidos do início. Essa estrutura de dados segue a lógica "primeiro a entrar, primeiro a sair", ou seja, o primeiro elemento adicionado à fila é o primeiro a ser removido.

As operações básicas em uma fila são a inserção (também chamada de enfileiramento ou push) de um elemento no final da fila e a remoção (também chamada de desenfileiramento ou pop) do elemento no início da fila. Outra operação comum em filas é a verificação do elemento no início da fila sem removê-lo (também chamada de front).

As filas podem ser implementadas em diversas linguagens de programação, usando estruturas de dados como arrays ou listas encadeadas. A escolha da implementação depende das necessidades específicas do algoritmo em questão.

Lembrete

Uma pilha é uma estrutura de dados em que os elementos são adicionados e removidos somente na extremidade oposta. Já uma fila é uma estrutura de dados em que os elementos são adicionados ao final e removidos do início.

Existem algumas limitações na busca em largura. Uma delas é que ela pode ser ineficiente em problemas de busca em que há muitos estados possíveis, pois explorará todos os estados, mesmo se a maioria deles não for relevante para a solução do problema. Além disso, ela pode ser ineficiente em problemas de busca com muitos nós profundos, pois precisará explorar todos os nós em cada profundidade antes de se aprofundar.

Apesar dessas limitações, a busca em largura é uma técnica útil para resolver muitos problemas de busca. Ela é simples de implementar e geralmente fornece soluções precisas e confiáveis. Também é uma boa escolha quando outros algoritmos de busca, como a busca em profundidade e a busca A*, não são aplicáveis ou não são eficientes.

A figura a seguir apresenta a função BUSCA-EM-LARGURA em um grafo:

```
função BUSCA-EM-LARGURA(problema) retorna uma solução ou falha
  nó ← um nó com ESTADO = problema.ESTADO-INICIAL, CUSTO-DE-CAMINHO = 0
  se problema.TESTE-DE-OBJETIVO(nó.ESTADO) senão retorne SOLUÇÃO(nó),
  borda ← uma fila FIFO com nó como elemento único
  explorado ← conjunto vazio
  repita
    se VAZIO?(borda), então retorne falha
    nó ← POP(borda) / * escolhe o nó mais raso na borda */
    adicione nó.ESTADO para explorado
    para cada ação em problema.AÇÕES(nó.ESTADO) faça
      filho ← NÓ-FILHO(problema, nó, ação),
      se (filho.ESTADO)não está em explorado ou borda então
        se problema.TESTE-DE-OBJETIVO(filho.ESTADO) então retorne SOLUÇÃO(filho)
        borda ← INSIRA(filho, borda)
```

Figura 15 – Busca em largura em um grafo

Fonte: Russell e Norvig (2013, p. 70).

A figura a seguir mostra a busca em largura em uma árvore binária simples:

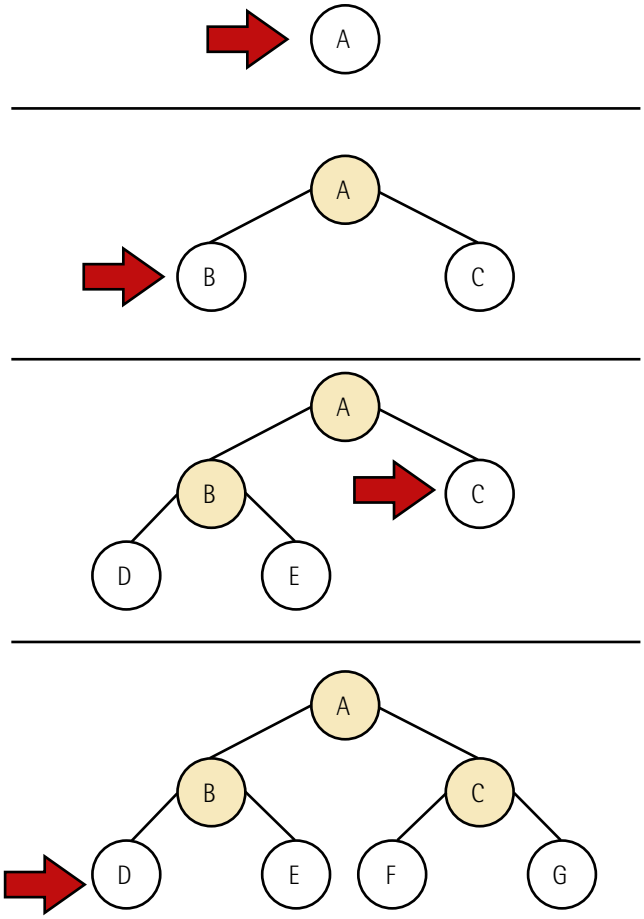


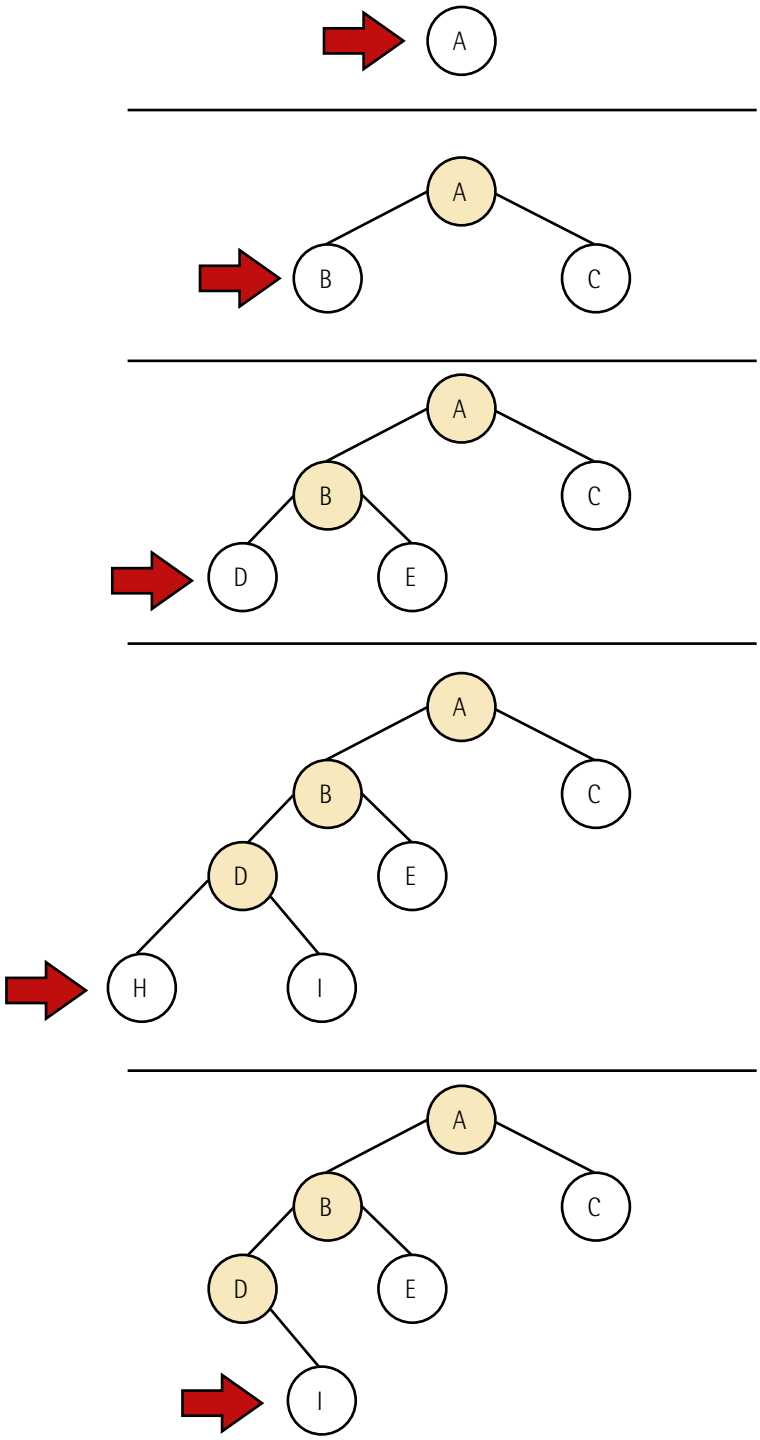
Figura 16 – Busca em largura em uma árvore binária simples.
Em cada fase, o próximo nó a ser expandido é indicado por um marcador

3.4 Busca em profundidade e profundidade limitada

A busca em profundidade é uma técnica de IA que é usada para resolver problemas de busca em grafos e árvores. O objetivo é percorrer todos os nós de um grafo ou árvore, explorando completamente cada ramo antes de passar para o próximo. É uma técnica recursiva que segue uma pré-ordem e é frequentemente comparada com a busca em largura, que busca através de todos os nós de forma sistemática e uniforme.

A busca em profundidade começa no nó inicial e adiciona todos os nós adjacentes à pilha. Ela então continua visitando cada nó na pilha até encontrar a solução desejada ou determinar que não há solução. Essa técnica é usada para resolver problemas como caminhos mínimos, labirintos e problemas de jogo como xadrez, go e sudoku.

A figura a seguir mostra a busca em profundidade em uma árvore binária simples:



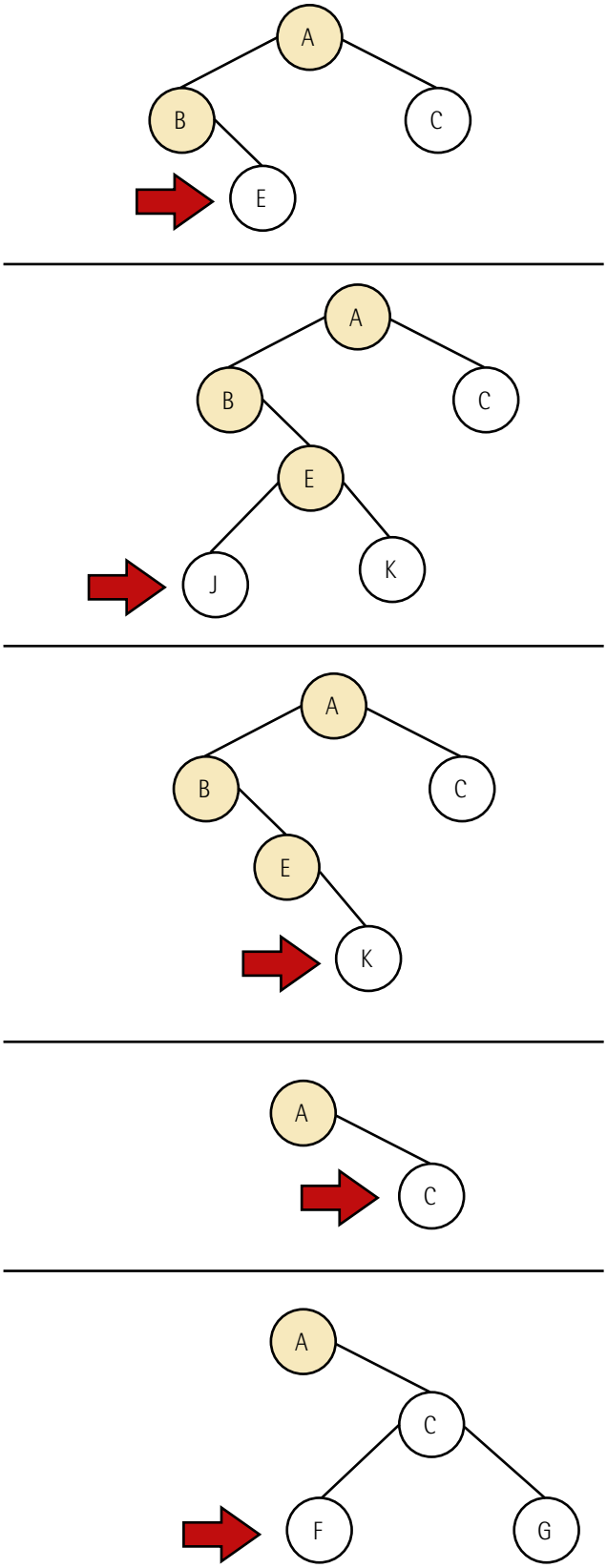


Figura 17 – Busca em profundidade em uma árvore binária.
Os nós explorados sem descendentes na borda são removidos da memória

A busca em profundidade tem algumas vantagens sobre a busca em largura. Por exemplo, ela pode ser mais eficiente em espaços de busca maiores e pode encontrar soluções mais curtas em problemas de caminhos mínimos. No entanto, essa técnica também tem algumas desvantagens, como a possibilidade de entrar em looping infinito e não garantir que todos os nós sejam visitados.

Além disso, a busca em profundidade tem algumas variações, como a busca em profundidade limitada e a busca em profundidade iterativa (conhecido também por iterative deepening A* – IDA*). A busca em profundidade limitada adiciona uma profundidade máxima para evitar looping infinito, enquanto a IDA* combina a busca em profundidade com a heurística A* para encontrar soluções mais rápidas.

A busca em profundidade limitada é uma variação da técnica de busca em profundidade em inteligência artificial (IA) que adiciona uma profundidade máxima para evitar looping infinito. A busca em profundidade tradicional tem como objetivo percorrer todos os nós de um grafo ou árvore, explorando completamente cada ramo antes de passar para o próximo, mas essa técnica pode entrar em looping infinito se não houver um mecanismo para evitar isso. A busca em profundidade limitada adiciona esse mecanismo, limitando a profundidade máxima da busca. A figura a seguir apresenta uma implementação recursiva da busca em árvore de profundidade limitada.

A busca em profundidade limitada começa no nó inicial e adiciona todos os nós adjacentes à pilha. Ele então continua visitando cada nó na pilha até encontrar a solução desejada ou atingir a profundidade máxima estabelecida. Se a solução não for encontrada, o algoritmo retrocede e tenta novamente a partir de um nó anterior, até que a solução seja encontrada ou todas as possibilidades sejam esgotadas. Ela é menos propensa a entrar em looping infinito do que a busca em profundidade tradicional, mas pode não garantir que se encontre a solução ótima, pois pode haver soluções melhores além da profundidade estabelecida.

```
função BUSCA-EM-PROFUNDIDADE-LIMITADA(problema, limite) retorna uma solução ou falha/corte
retornar BPL-RECURSIVA (CRIAR-NÓ(problema, ESTADO-INICIAL), problema, limite)
função BPL-RECURSIVA(nó, problema, limite) retorna uma solução ou falha/corte
  se problema. TESTAR-OBJETIVO (nó.ESTADO) então, retorna SOLUÇÃO (nó)
  se não se limite = 0 então retorna corte
  senão
    corte_ocorreu? ← falso para cada ação no problema.AÇÕES(nó.ESTADO) faça
      filho ← NÓ-FILHO (problema, nó, ação)
      resultado ← BPL-RECURSIVA (criança, problema limite – 1)
      se resultado = corte  então corte_ocorreu? ← verdadeiro
      senão se resultado ≠ falha então retorna resultado
    se corte_ocorreu? então retorna corte senão retorna falha
```

Figura 18 – Uma implementação recursiva da busca em árvore de profundidade limitada

Fonte: Russell e Norvig (2013, p. 74).

3.5 Busca em profundidade iterativa

O algoritmo de busca em profundidade iterativa (ou IDDFS, do inglês iterative deepening depth-first search) é uma estratégia de busca em grafos que combina as vantagens da busca em profundidade com a eficiência da busca em largura.

A ideia básica do algoritmo é executar sucessivas buscas em profundidade, limitando a profundidade máxima em cada busca, até encontrar o nó desejado. A profundidade máxima é aumentada a cada iteração, e o algoritmo para assim que encontra o nó desejado ou quando alcança a profundidade máxima estabelecida.

O IDDFS funciona da seguinte maneira:

1 – Inicia-se a busca a partir do nó raiz com profundidade máxima 0.

2 – Se o nó atual é o nó de destino, a busca termina e o caminho é retornado.

3 – Se a profundidade máxima foi atingida e o nó de destino não foi encontrado, retorna-se para o nó anterior.

4 – Caso contrário, expande-se o nó atual e adiciona-se seus filhos à fronteira.

5 – Repete-se o processo para o próximo nó na fronteira.

A cada iteração, a profundidade máxima é incrementada em 1, e a busca é reiniciada a partir do nó raiz. Como o algoritmo usa a estratégia de busca em profundidade, ele consome menos memória do que a busca em largura, já que só precisa armazenar a fronteira atual. Por outro lado, como ele itera várias vezes, pode ser menos eficiente do que outras estratégias de busca em alguns casos. A figura a seguir apresenta o algoritmo de busca de aprofundamento iterativo:

```
função BUSCA-DE-APROFUNDAMENTO-ITERATIVO(problema) retorna uma solução ou falha
para profundidade = 0 até ∞ faça
resultado ← BUSCA-EM-PROFUNDIDADE-LIMITADA(problema, profundidade)
se resultado ≠ corte então retornar resultado
```

Figura 19 – Algoritmo de busca de aprofundamento iterativo que aplica repetidamente a busca em profundidade limitada com limites crescentes. Ele termina quando uma solução é encontrada ou se a busca em profundidade limitada devolve falha, indicando que não existe nenhuma solução

Fonte: Russell e Norvig (2013, p. 75).

O IDDFS é útil em problemas em que o grafo é grande e não há informações sobre a profundidade do caminho mais curto entre o nó inicial e o nó de destino. Além disso, é uma alternativa ao algoritmo

de busca em largura, quando a profundidade do caminho é desconhecida, mas se deseja manter o consumo de memória em um nível razoável. A figura a seguir apresenta quatro iterações de busca de aprofundamento iterativo em uma árvore binária.

Uma das principais vantagens do IDDFS é que ele sempre encontra a solução mais rasa possível, em relação a outros algoritmos de busca em profundidade. Isso ocorre porque, em cada iteração, ele expande todos os nós que estão na profundidade máxima atual, antes de aumentá-la.

Por exemplo, suponha que estamos buscando o caminho mais curto entre o nó A e o nó Z em um grafo, usando o IDDFS com uma profundidade máxima de 3. Se o caminho mais curto entre A e Z tiver profundidade 4, o IDDFS não irá encontrá-lo na primeira iteração, pois só irá expandir nós com profundidade máxima 3. Entretanto, na segunda iteração, ele irá expandir todos os nós com profundidade 3 e, portanto, encontrará o caminho mais curto.

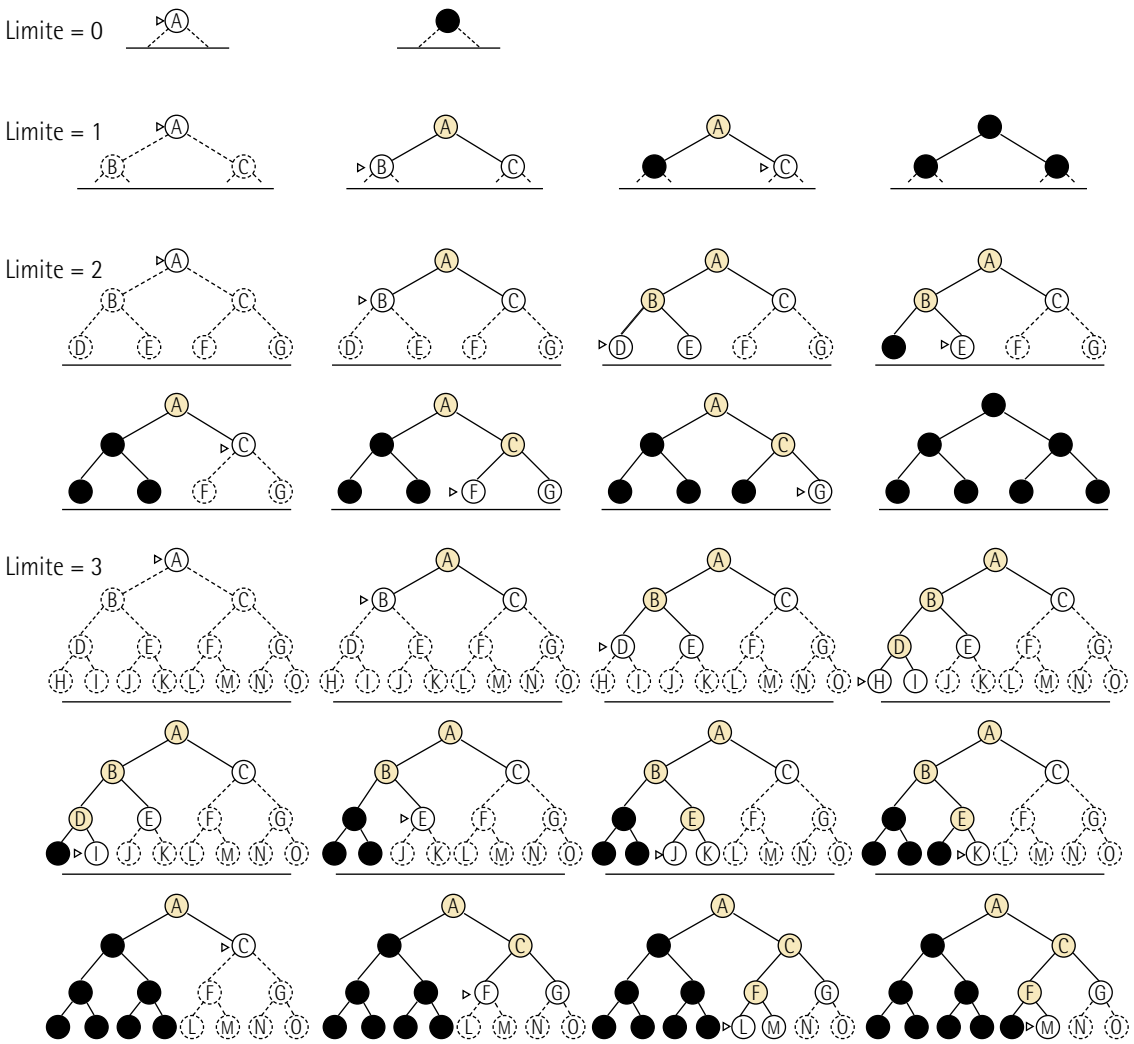


Figura 20 – Quatro iterações de busca de aprofundamento iterativo em uma árvore binária

Fonte: Russell e Norvig (2013, p. 77).

Uma possível desvantagem do IDDFS é que ele pode expandir os mesmos nós várias vezes, em iterações diferentes. Entretanto, em muitos casos, a economia de memória compensa essa desvantagem.

Outra desvantagem é que o IDDFS não é adequado para grafos com profundidades muito grandes, já que o número de iterações necessárias pode se tornar muito grande. Nesse caso, outras estratégias de busca podem ser mais eficientes.

4 ALGORITMOS DE BUSCA INFORMADA

Os algoritmos de busca informada (também conhecidos como algoritmos de busca heurística) são uma classe de algoritmos de busca em grafos que utilizam informações adicionais para guiar a busca em direção ao objetivo. Essas informações podem ser heurísticas, ou seja, estimativas do quão perto um nó está do objetivo, ou informações sobre a estrutura do grafo que está sendo pesquisado.

Os algoritmos de busca informada são úteis em problemas em que o grafo é grande e a busca em profundidade ou busca em largura não são eficientes o suficiente. Esses algoritmos são capazes de guiar a busca para as áreas mais promissoras do grafo, reduzindo o número de nós visitados e, portanto, melhorando a eficiência da busca.

4.1 Heurísticas

As heurísticas são funções que fornecem uma estimativa do custo de chegar ao objetivo a partir de um estado dado. Elas são amplamente utilizadas na IA para guiar a busca em problemas com espaços de estado grandes e complexos, como planejamento, roteamento, alocação de recursos e outras tarefas que envolvem a busca de uma solução ótima ou subótima.

As heurísticas são usadas em algoritmos de busca informada, como A* e IDA*, para calcular o custo f do nó, que é a soma do custo g (custo até o nó) e do custo h (estimativa do custo até o objetivo). Um algoritmo de busca informado com uma heurística precisa é capaz de encontrar uma solução ótima em tempo polinomial.

Existem diferentes tipos de heurísticas, como heurísticas admissíveis e consistentes. Uma heurística é admissível se ela nunca subestima o custo real de chegar ao objetivo. Uma heurística é consistente se ela cumpre a desigualdade de triangular, ou seja, o custo de chegar a um nó filho nunca é maior que o custo de chegar ao nó pai mais o custo de chegar ao filho.

As heurísticas são geralmente específicas para cada problema, e podem ser construídas com base em conhecimento humano sobre o problema, ou aprendidas com dados. Alguns exemplos de heurísticas incluem a distância Manhattan para problemas de roteamento em um grid, ou a contagem de peças fora do lugar para problemas de jogo, como em um cubo mágico.

A heurística de distância Manhattan é uma técnica utilizada em algoritmos de busca informada, como o A* para estimar o custo de um caminho de um nó até o objetivo em um grafo. Essa heurística leva em conta apenas as posições dos nós no grafo, ignorando a existência de obstáculos ou outras informações adicionais.

A distância Manhattan é chamada assim porque se assemelha à distância que seria percorrida por um táxi em uma cidade com ruas dispostas em um padrão de grade. Para calcular a distância Manhattan entre dois nós, basta somar as diferenças absolutas das coordenadas x e y dos nós. Por exemplo, se temos dois nós A e B com coordenadas (x1, y1) e (x2, y2), respectivamente, a distância Manhattan entre eles é dada pela fórmula:

$$\text{distânciaManhattan}(A, B) = |x1 - x2| + |y1 - y2|$$

Essa heurística é muito utilizada em problemas de busca em labirintos ou jogos em que o objetivo é se mover de um ponto inicial para um ponto final. Nesses casos, a distância Manhattan pode ser usada para estimar o custo do caminho mais curto para alcançar o objetivo, assumindo que só é possível se mover nas direções horizontais e verticais.

Embora a distância Manhattan seja uma heurística simples, ela pode ser muito eficaz na prática, especialmente em problemas em que o espaço de busca é relativamente pequeno. No entanto, em casos mais complexos, outras heurísticas mais avançadas, que levam em conta informações adicionais sobre o ambiente, podem ser necessárias para obter uma estimativa mais precisa do custo do caminho mais curto.

4.2 Busca gulosa

A busca gulosa é um algoritmo de busca heurística que é usado para encontrar soluções aproximadas em problemas de otimização em que o objetivo é encontrar a melhor solução possível entre muitas possíveis soluções. Nesse algoritmo, a cada passo, é feita a escolha que parece ser a melhor no momento, sem levar em consideração as consequências futuras dessa escolha.

Na busca gulosa, o algoritmo avalia todos os possíveis caminhos a partir do estado atual e escolhe o caminho que parece mais promissor, com base em uma heurística específica. A heurística é usada para estimar a distância ou o custo do caminho para o objetivo final, e é essa estimativa que é usada para determinar qual caminho seguir.

O objetivo da busca gulosa é encontrar uma solução rapidamente, mesmo que ela não seja necessariamente a melhor solução possível. Em outras palavras, a busca gulosa é um algoritmo que busca uma solução que seja "suficientemente boa", mas não necessariamente ótima.

A principal vantagem da busca gulosa é que ela é muito rápida e eficiente em encontrar soluções para problemas de grande escala. No entanto, a desvantagem é que a solução encontrada pode não ser a melhor possível. Isso ocorre porque a heurística usada na busca gulosa pode levar a soluções que ficam presas em um mínimo local, em vez de alcançar o melhor resultado global.

Além disso, a busca gulosa pode ser facilmente enganada por casos em que a heurística é enganosa ou não é adequada para o problema. Por exemplo, se a heurística usada na busca gulosa for muito simples, ela pode não ser capaz de encontrar a solução ideal para um problema complexo.

Podemos utilizar esse algoritmo em uma ampla variedade de problemas de otimização, como problemas de roteamento em redes, problemas de programação linear, problemas de encaixe, problemas de escalonamento e muitos outros.

Em problemas de roteamento em redes, por exemplo, a busca gulosa pode ser usada para encontrar a melhor rota entre dois pontos em uma rede, levando em consideração fatores como o comprimento da rota e o número de conexões envolvidas. A heurística usada na busca gulosa pode ser baseada em fatores como o número de saltos na rede ou a largura de banda disponível em cada conexão.

Na programação linear, a busca gulosa pode ser usada para encontrar a solução mais viável para um conjunto de restrições lineares, levando em consideração a função objetivo a ser otimizada. A heurística usada na busca gulosa pode ser baseada em fatores como a magnitude dos coeficientes da função objetivo ou o número de variáveis envolvidas no problema.

Em problemas de encaixe, podemos utilizar a busca gulosa para encontrar a melhor maneira de encaixar um conjunto de objetos em um espaço limitado, levando em consideração fatores como o tamanho dos objetos e a geometria do espaço disponível. A heurística usada na busca gulosa pode ser baseada em fatores como a área ocupada por cada objeto ou a distância entre os objetos.

Como exemplo prático, vamos utilizar o problema de roteamento na Romênia e a heurística de distância em linha reta (hDLR). O objetivo é chegar em Bucareste, então precisaremos saber as distâncias em linha reta até Bucareste. No quadro a seguir podemos observar hDLR (Em(Arad)) = 366. A heurística é útil, pois a hDLR está correlacionada com as distâncias reais da estrada.

Quadro 4 – Valores de hDLR: distâncias em linha reta para Bucareste

Arad	366
Bucareste	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Fonte: Russell e Norvig (2013, p. 79).

A figura a seguir mostra como funciona o algoritmo de uma busca gulosa utilizando a melhor escolha de hDLR para encontrar um caminho de Arad para Bucareste:

- o primeiro nó expandido a partir de Arad será Sibiu (mais perto de Bucareste do que Zerind ou Timisoara);
- o próximo nó a ser expandido será Fagaras devido a sua proximidade;
- Fagaras vai gerar Bucareste, que é o objetivo.

Para esse problema, a busca gulosa de melhor escolha utilizando hDLR encontra uma solução. Não é a solução ótima, o caminho via Sibiu e Fagaras para Bucareste é 32 quilômetros mais longo que o caminho através de Rimnicu Vilcea e Pitesti. Podemos ver que o algoritmo é "ambicioso"; a cada passo ele busca chegar o mais próximo do objetivo.

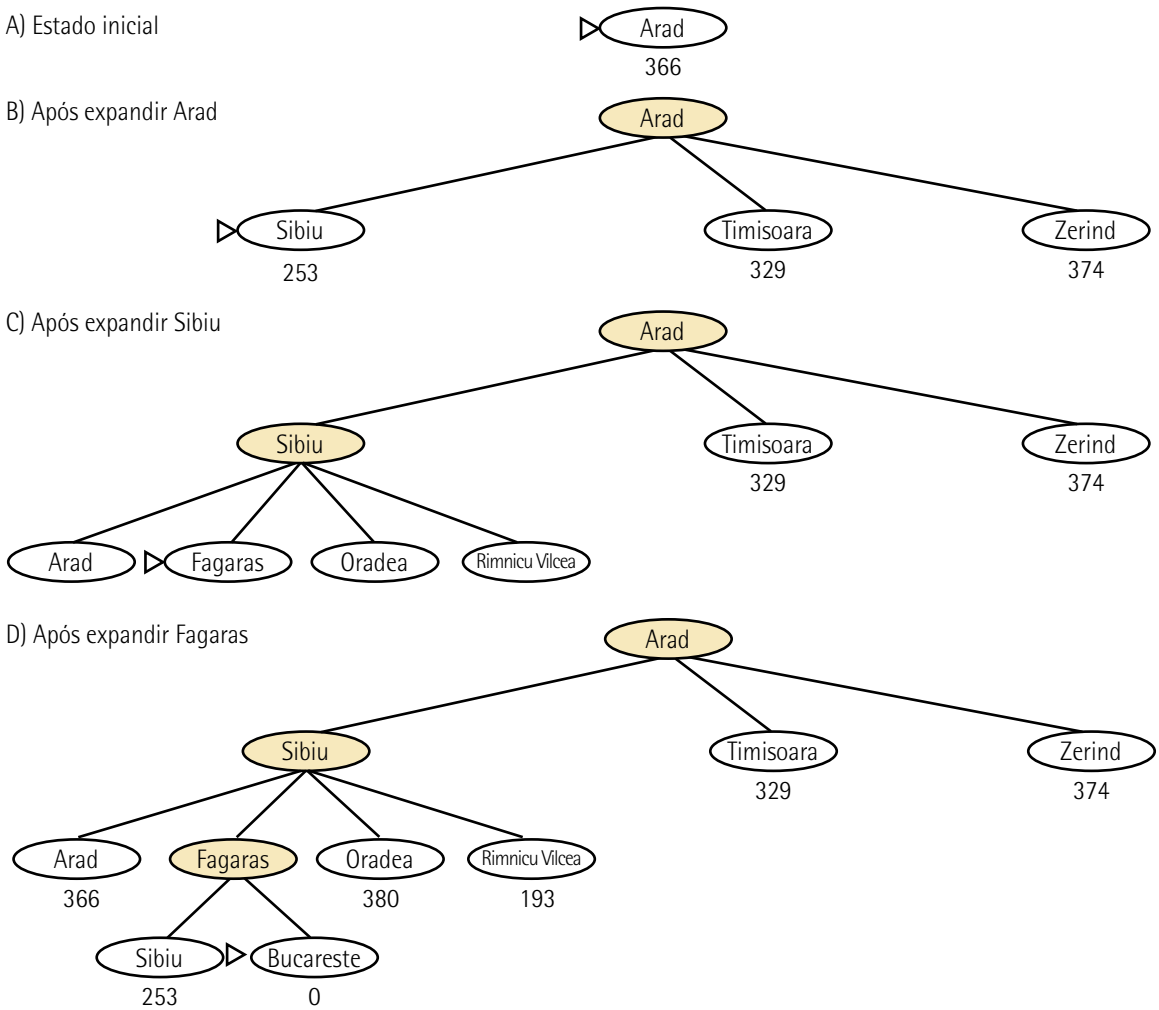


Figura 21 – Etapas de uma busca gulosa de melhor escolha em árvore para Bucareste com a heurística de distância em linha reta hDLR. Os nós são rotulados com os seus valores h

Fonte: Russell e Norvig (2013, p. 80).

4.3 Busca A*

A busca A* é um algoritmo de busca informada que utiliza uma heurística, uma função que estima o custo de chegar ao objetivo a partir de um estado dado, para guiar a busca. Ele é amplamente utilizado em tarefas de planejamento, roteamento, alocação de recursos e outras tarefas que envolvem a busca de uma solução ótima.

A* funciona expandindo os nós do espaço de estado de acordo com o valor da função f, que é a soma do custo g (custo até o nó) e do custo h (estimativa do custo até o objetivo). O nó com o menor valor de f é selecionado para ser expandido.

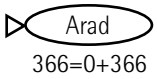
A* utiliza uma estrutura de dados, como uma fila de prioridade, para armazenar os nós expandidos e ordená-los de acordo com o valor de f. Assim, sempre que um novo nó é expandido, ele é adicionado à fila de prioridade e o nó com o menor valor de f é selecionado para ser expandido.

f (n) = custo estimado da solução de menor custo através de n.

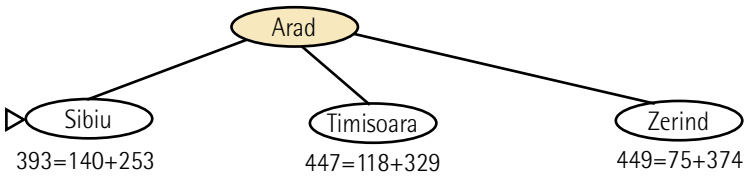
f (n) = g(n) + h(n).

A complexidade de tempo de A* é O(b^m), onde b é o branching factor (número médio de filhos de cada nó), m é o número de nós no caminho da solução e O(b*m) é sua complexidade de espaço. Uma das vantagens de A* é que ele é garantido para encontrar a solução ótima se a heurística é admissível e consistente.

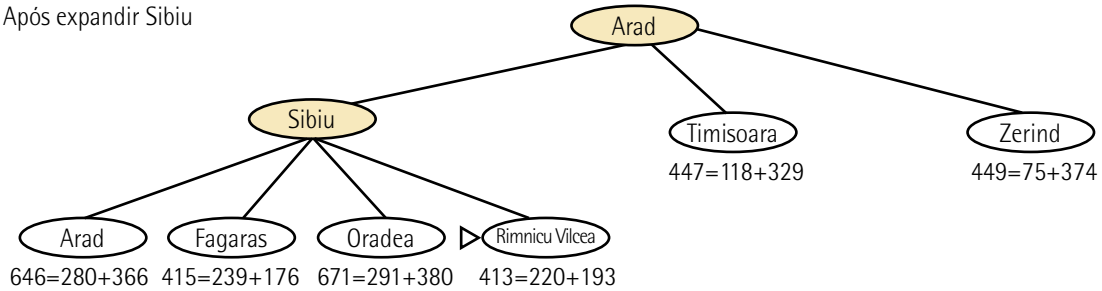
A) Estado inicial



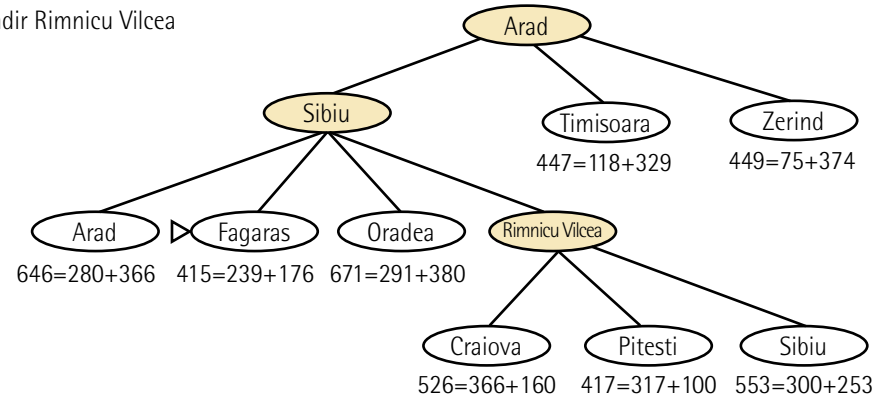
B) Após expandir Arad



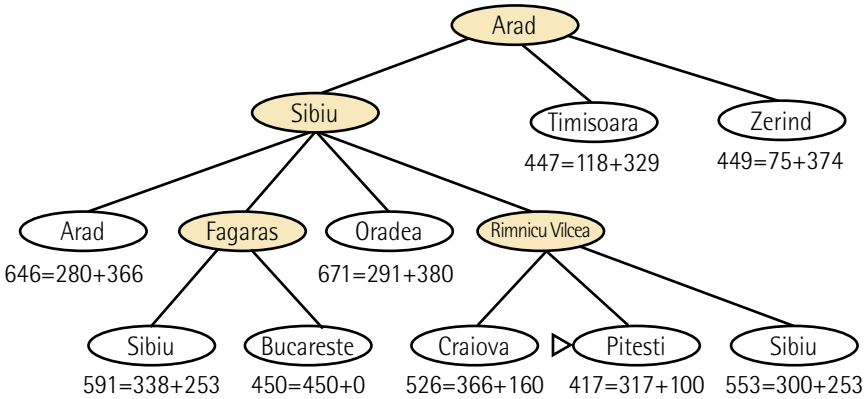
C) Após expandir Sibiu



D) Após expandir Rimnicu Vilcea



E) Após expandir Fagaras



F) Após expandir Fagaras

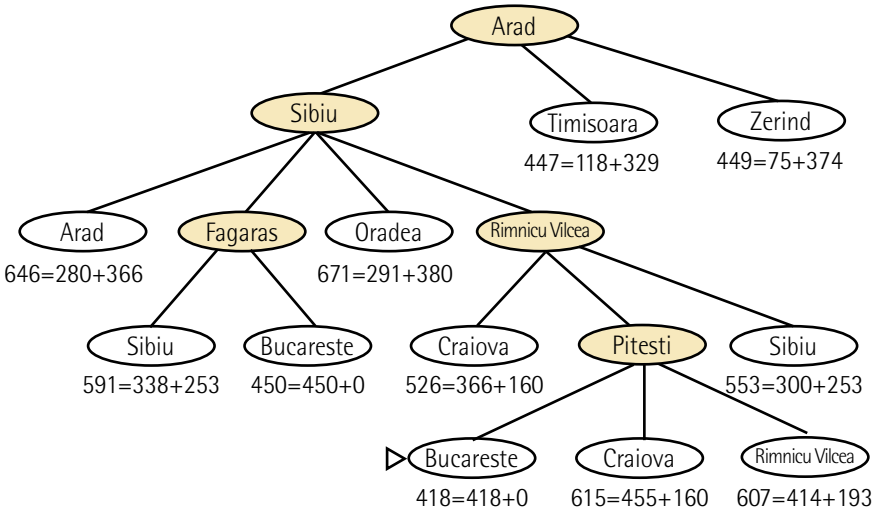


Figura 22 – Etapas em uma busca A* para Bucareste. Nós são rotulados com f = g + h.
Os valores de h são as distâncias em linha reta para Bucareste

Fonte: Russell e Norvig (2013, p. 81).

4.4 Busca best-first

O algoritmo best-first é um algoritmo de busca heurística usado para encontrar caminhos em grafos. Ele é um tipo de algoritmo guloso (greedy), pois escolhe o próximo nó para explorar com base na heurística que indica qual nó parece ser o "melhor" em direção ao objetivo.

Esse algoritmo começa explorando o nó inicial e, em seguida, escolhe o próximo nó a ser explorado com base na heurística escolhida. Essa heurística pode ser uma estimativa da distância do nó atual até o objetivo, por exemplo. O algoritmo seleciona o nó que parece estar mais próximo do objetivo, de acordo com a heurística, e continua a explorar os nós adjacentes a ele.

Ele não considera o custo total do caminho percorrido até o nó atual, mas apenas a heurística escolhida. Isso pode levar a soluções subótimas em alguns casos, pois o algoritmo pode se concentrar em um caminho que parece promissor de acordo com a heurística, mas que pode ser mais caro do que outro caminho mais longo que leva a uma solução melhor.

No entanto, o algoritmo best-first é eficiente em termos de tempo de execução, pois evita explorar caminhos que parecem improváveis de levar ao objetivo. Isso é especialmente útil em grafos muito grandes ou complexos, em que explorar todos os caminhos seria impraticável ou impossível.

Ele pode ser usado em várias aplicações, como planejamento de rotas, jogos de quebra-cabeça e sistemas de recomendação. No entanto, deve ser lembrado que esse algoritmo pode levar a soluções subótimas em alguns casos e pode não encontrar a solução ótima em todos os casos.

Uma das variações do algoritmo best-first é o algoritmo A*, que combina a heurística com o custo real do caminho percorrido até o nó atual. O algoritmo A* é geralmente considerado mais eficiente que o best-first, pois usa a heurística para evitar explorar caminhos que parecem improváveis de levar ao objetivo, mas também leva em consideração o custo real do caminho percorrido até o nó atual.

Outra variação do algoritmo best-first é o algoritmo hill climbing, que escolhe o próximo nó a ser explorado com base na heurística e segue sempre em direção ao nó que parece estar mais próximo do objetivo, sem considerar os nós já explorados anteriormente. Esse algoritmo é especialmente útil em casos em que a heurística é precisa e o objetivo é fácil de alcançar, mas pode levar a soluções subótimas ou ficar preso em um mínimo local.

A complexidade de tempo da busca best-first é difícil de ser analisada, pois depende da heurística utilizada e do número de nós expandidos antes de encontrar a solução. No entanto, geralmente é considerado como sendo menor ou igual ao algoritmo A* que é O(b^m), onde b é o branching factor (número médio de filhos de cada nó) e m é o número de nós no caminho da solução. Consideramos sua complexidade de espaço O(b*m).

4.5 Busca IDA*

A busca IDA* (iterative deepening A*) é um algoritmo de busca informada que combina a busca em profundidade com a busca A*. Ele funciona expandindo os nós do espaço de estado em ordem de profundidade. Em cada iteração, um limite de custo é estabelecido e somente os nós com custo menor que o limite são expandidos. O limite é aumentado a cada iteração até que a solução ótima seja encontrada.

A IDA* utiliza uma heurística, uma função que estima o custo de chegar ao objetivo a partir de um estado dado, para guiar a busca. A heurística é usada para calcular o custo f do nó, que é a soma do custo g (custo até o nó) e do custo h (estimativa do custo até o objetivo). O limite é estabelecido como o menor valor f encontrado entre os nós não expandidos.

A complexidade de tempo de IDA* é O(b^d) onde b é o branching factor (número médio de filhos de cada nó), d é a profundidade da solução e O(bd) é sua complexidade de espaço.

Uma das vantagens de IDA* é que ele é eficiente para lidar com problemas com espaços de estado muito grandes, pois não precisa armazenar todos os nós expandidos em memória, somente os nós que estão na fronteira da busca.

4.6 Busca recursiva best-first (RBFS)

A busca recursiva best-first (RBFS) é um algoritmo de busca informada que combina a busca recursiva com a busca best-first. RBFS funciona expandindo os nós do espaço de estado recursivamente, e, a cada chamada recursiva, ele seleciona o nó com a melhor avaliação heurística para ser expandido. A diferença entre o RBFS e outros algoritmos de busca best-first é que o RBFS não utiliza uma fila de prioridade

para armazenar os nós expandidos. Em vez disso, ele utiliza a chamada recursiva para organizar os nós. O algoritmo é mostrado na figura a seguir.

A heurística é usada para calcular o custo f do nó, que é a soma do custo g (custo até o nó) e do custo h (estimativa do custo até o objetivo). A complexidade de tempo de RBFS é difícil de ser analisada, pois depende da heurística utilizada e do número de nós expandidos antes de encontrar a solução. No entanto, geralmente é considerado melhor ou igual ao algoritmo A* que é O(b^m), onde b é o branching factor (número médio de filhos de cada nó), m é o número de nós no caminho da solução e O(b*m) é sua complexidade de espaço.

Uma das vantagens da busca recursiva best-first é que ele é capaz de encontrar soluções subótimas mais rapidamente do que outros algoritmos de busca informada, como A*. Ele também é capaz de lidar com problemas com espaços de estado muito grandes, pois não precisa armazenar todos os nós expandidos em memória, somente os nós que estão na fronteira da busca.

```
função BUSCA-RECURSIVA-PELA-MELHOR(problema) retorna uma solução ou falha
  retorna RBFS(problema, FAZ-NÓ(problema. ESTADO-INICIAL), ∞)
função RBFS (problema, nó, f_limite) retorna uma solução ou falha e um limite novo f_custo
  se problema. TESTE-OBJETIVO(nó.ESTADO) então retorne SOLUÇÃO (nó)
  sucessores ← [ ]
  para cada ação em problema. AÇÕES(nó.ESTADO) fazer
    adicionar NÓ-FILHO(problema, nó, ação) em sucessores
  se sucessores estiver vazio então retornar falha), ∞
  para cada s em sucessores fazer / * atualizar f com o valor da busca anterior, se houver */
    s.f ← max(s.g + s.h, nó.f))
  repita
    melhor ← valor f mais baixo do nó em sucessores
    se melhor.f > f_limite então retornar falha, melhor.f
    alternativa ← segundo valor f mais baixo entre sucessores
    resultado, melhor.f ← RBFS(problema, melhor, min(f_limite, alternativa)
    se resultado ≠ falha então retornar resultado
```

Figura 23 – Algoritmo para busca de melhor escolha recursiva

Fonte: Russell e Norvig (2013, p. 85).

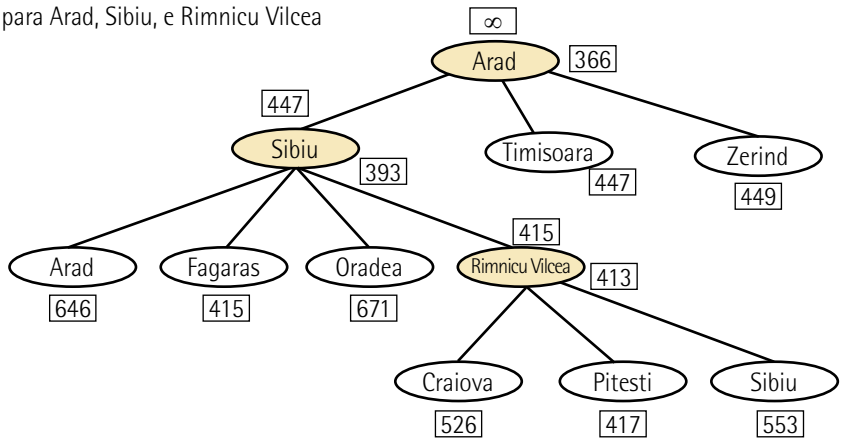
A figura a seguir apresenta as etapas de uma busca RBFS para a rota mais curta para Bucareste. O valor do f_limite para cada chamada recursiva é apresentado no topo do nó atual, e cada nó é rotulado com seu f-custo. De acordo com Russell e Norvig (2013, p. 85):

(a) O caminho via Rimnicu Vilcea é seguido até que a melhor folha atual (Pitesti) tenha um valor que é pior do que o melhor caminho alternativo (Fagaras).

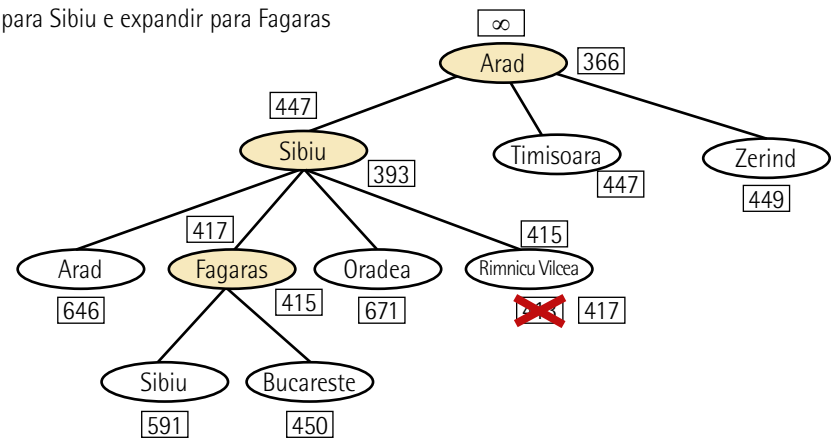
(b) A recursão reverte e o melhor valor da folha da subárvore esquecida (417) é copiado para Rimnicu Vilcea, então Fagaras é expandida, revelando um melhor valor da folha de 450.

(c) A recursão reverte e o melhor valor da folha da subárvore esquecida (450) é copiado para Fagaras, então Rimnicu Vilcea é expandida. Dessa vez, devido ao melhor caminho alternativo (através de Timisoara), custa pelo menos 447, e a expansão continua para Bucareste.

A) Após expandir para Arad, Sibiu, e Rimnicu Vilcea



B) Após reverter para Sibiu e expandir para Fagaras



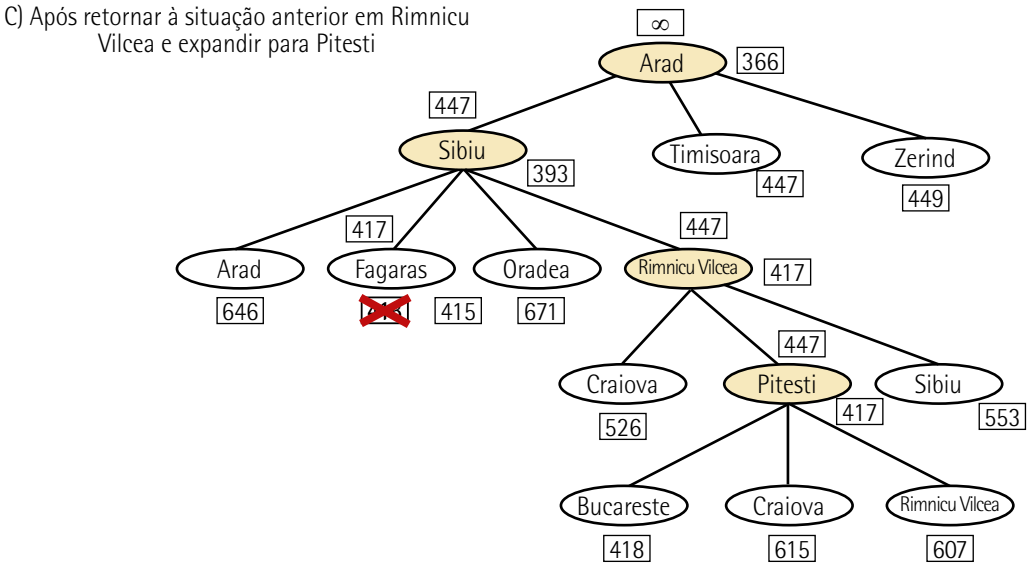


Figura 24 – Etapas de uma busca RBFS para a rota mais curta para Bucareste

Fonte: Russell e Norvig (2013, p. 85).

4.7 Busca escalada na montanha (hill climbing)

A busca escalada na montanha (hill climbing) é um algoritmo de busca local utilizado para encontrar a solução ótima ou subótima de um problema. A ideia básica do algoritmo é começar com uma solução inicial e, a cada iteração, gerar sucessores para a solução atual e escolher o sucessor que melhor se aproxima da solução ótima. A busca é interrompida quando uma solução local ótima é encontrada ou quando não há mais melhorias.

O nome **escalada na montanha** é uma analogia ao movimento que um alpinista faria ao subir uma montanha, pois ele sempre tenta subir em direção ao ponto mais alto e não necessariamente precisa saber qual é esse ponto.

O funcionamento básico do algoritmo hill climbing é o seguinte:

1. Gere um estado inicial aleatório ou escolha um estado inicial qualquer.
2. Calcule o valor da função objetivo para o estado atual.
3. Gere um conjunto de estados sucessores, realizando alterações no estado atual.
4. Avalie os valores da função objetivo para cada estado sucessor gerado.
5. Escolha o estado sucessor que apresente o maior valor de função objetivo e faça dele o novo estado atual.
6. Repita os passos 2 a 5 até que não haja mais estados sucessores que apresentem um valor de função objetivo maior do que o estado atual.

Para utilizar o algoritmo de busca escalada na montanha, é necessário definir uma função de avaliação para medir a quão boa é uma solução dada. Essa função deve ser uma função de otimização, na qual quanto maior o valor retornado, melhor é a solução.

A complexidade de tempo de hill climbing é O(d*n^2), onde d é a profundidade da busca, n é o número de sucessores gerados e O(d) é sua complexidade de espaço.

O algoritmo de busca escalada na montanha é simples de implementar e tem um bom desempenho para problemas pequenos, mas pode se sair mal em problemas maiores ou com muitos mínimos locais, pois ele pode ficar preso em um mínimo local e não encontrar a solução global. Além disso, se a função de avaliação não for bem escolhida, a busca pode ser ineficiente e nunca encontrar a solução ótima.

4.8 Algoritmos genéticos

Os algoritmos genéticos são uma técnica de busca inspirada na evolução biológica, amplamente utilizada em inteligência artificial e otimização. Eles são utilizados para encontrar soluções ótimas ou subótimas para problemas complexos e de alta dimensionalidade que são difíceis de serem resolvidos com algoritmos convencionais.

Os algoritmos genéticos funcionam criando uma população de soluções candidatas e, através de operações de seleção, cruzamento e mutação, evoluindo essa população ao longo de várias gerações. A cada geração, as soluções são avaliadas usando uma função de fitness e as melhores soluções são selecionadas para gerar a próxima geração. O processo é repetido até que uma solução satisfatória seja encontrada ou até que o algoritmo atinja o número máximo de gerações.

Os algoritmos genéticos são altamente parametrizáveis, permitindo ajustar coisas como a taxa de cruzamento, a taxa de mutação, o tamanho da população, entre outras coisas, para adaptar o algoritmo para o problema específico.

A complexidade de tempo de um algoritmo genético é geralmente expressa em termos de número de gerações e tamanho da população, e é normalmente O(ngpf), onde n é o número de gerações, p é o tamanho da população, f é o tempo de avaliação da função de fitness e O(p) é sua complexidade de espaço.

Embora os algoritmos genéticos possam ser computacionalmente intensivos, eles são capazes de lidar com problemas altamente complexos e de alta dimensionalidade.

Vamos utilizar como exemplo o problema das oito rainhas, um problema clássico de posicionamento de peças em um tabuleiro de xadrez. O objetivo é posicionar oito rainhas em um tabuleiro de xadrez de 8 × 8, de tal forma que nenhuma rainha possa atacar outra. Isso significa que nenhuma rainha pode estar na mesma linha, coluna ou diagonal de outra rainha.

O problema das oito rainhas é um exemplo de problema de posicionamento ou de colocação de objetos, e é um desafio popular em ciência da computação, teoria dos grafos e inteligência artificial. O problema tem solução, e a solução única é conhecida desde o século XIX.

Existem diversas abordagens para resolver o problema das oito rainhas, incluindo algoritmos de busca, algoritmos genéticos e outras técnicas. Uma abordagem comum é utilizar o algoritmo de busca em profundidade, que explora todas as possíveis configurações de posicionamento das oito rainhas, avaliando cada configuração para verificar se ela é uma solução válida.

O problema das oito rainhas é um exemplo simples de problema de colocação de objetos, mas é um desafio interessante em termos de lógica e computação. Além disso, o problema tem aplicações práticas em áreas como o projeto de circuitos integrados, a alocação de recursos em sistemas de informação e a otimização de rotas em logística.

A fim de resolver o problema das oito rainhas com algoritmos genéticos, podemos seguir os seguintes passos:

1. Codificar as soluções como indivíduos: cada indivíduo em nosso espaço de busca será uma possível configuração de posicionamento das oito rainhas em um tabuleiro de xadrez. Podemos codificar as soluções como uma lista de oito números, onde cada número representa a posição de uma rainha em uma linha diferente do tabuleiro.

2. Gerar uma população inicial: para iniciar nosso algoritmo genético, precisamos gerar uma população inicial de indivíduos aleatórios. Podemos gerar indivíduos aleatórios com uma distribuição uniforme de valores para cada posição das rainhas.

3. Avaliar a aptidão dos indivíduos: para avaliar a aptidão de cada indivíduo, precisamos verificar se a configuração de posicionamento das oito rainhas satisfaz as restrições do problema. Podemos contar o número de pares de rainhas que estão na mesma linha, coluna ou diagonal e usar essa contagem como medida de aptidão. Indivíduos com menor número de pares de rainhas em conflito terão maior aptidão.

4. Selecionar indivíduos para reprodução: para selecionar indivíduos para reprodução, podemos usar técnicas de seleção como a roleta ou torneio. Indivíduos com maior aptidão terão maior probabilidade de serem selecionados.

5. Cruzar indivíduos selecionados: para cruzar indivíduos selecionados, podemos usar técnicas de recombinação como o crossover uniforme. Isso envolve escolher um ponto de corte aleatório na lista de posições de rainhas de cada indivíduo e trocar as subsequências antes e depois desse ponto de corte para criar dois novos indivíduos.

6. Mutar indivíduos resultantes do crossover: para evitar que nosso algoritmo genético fique preso em mínimos locais, podemos aplicar mutações aleatórias aos indivíduos resultantes do crossover. Isso envolve escolher aleatoriamente uma posição no cromossomo e alterar seu valor, com uma pequena probabilidade.

7. Avaliar a aptidão dos novos indivíduos: depois de cruzar e mutar os indivíduos selecionados, precisamos avaliar novamente a aptidão dos novos indivíduos resultantes. Podemos usar a mesma função de aptidão usada na etapa 3.

8. Repetir os passos 4 a 7 até atingir uma condição de parada: repita os passos 4 a 7 até atingir uma condição de parada, como encontrar uma solução válida ou obter um número máximo de gerações sem encontrar uma solução válida.

O algoritmo genético é uma técnica poderosa e versátil para resolver o problema das oito rainhas e outros problemas de otimização combinatória.

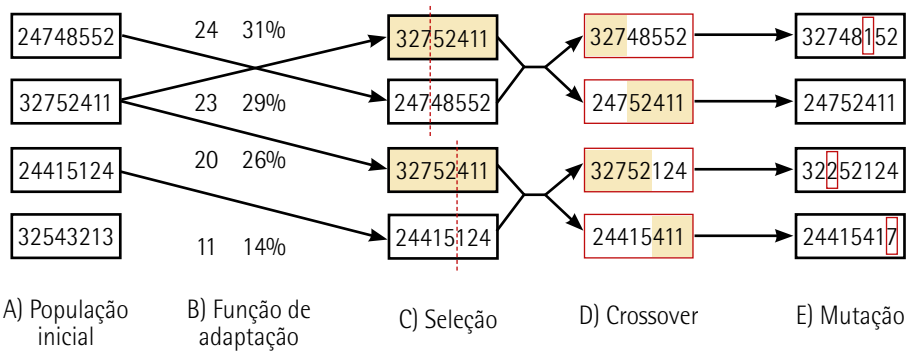


Figura 25 – O algoritmo genético, ilustrado por sequências de dígitos que representam os estados das oito rainhas. A população inicial em (a) é classificada pela função de adaptação em (b), resultando em pares de correspondência em (c). Eles produzem descendentes em (d), sujeitos à mutação em (e)

Fonte: Russell e Norvig (2013, p. 107).

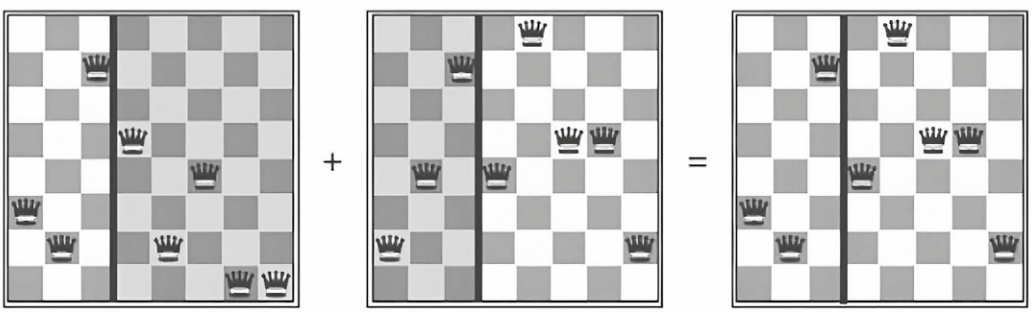


Figura 26 – Os estados das oito rainhas correspondentes aos dois primeiros pais na figura 25, em (c), e à primeira descendência da figura 25, em (d). As colunas sombreadas foram perdidas na etapa de cruzamento, e as colunas não sombreadas foram mantidas

Fonte: Russell e Norvig (2013, p. 109).

```
função ALGORITMO-GENÉTICO(população, FN-ADAPTA) retorna um indivíduo
  entradas: população, um conjunto de indivíduos
            FN-ADAPTA, uma função que mede a adaptação de um indivíduo
  repita
    nova_população ← conjunto vazio
    para i = 1 até TAMANHO(população) faça
      x ← SELEÇÃO-ALEATÓRIA(população, FN-ADAPTA)
      y ← SELEÇÃO-ALEATÓRIA(população, FN-ADAPTA)
      filho ← REPRODUZ(x, y)
      se (pequena probabilidade aleatória) então filho ← MUTAÇÃO(filho)
      adicionar filho a nova_população
      população ← nova_população
  até algum indivíduo estar adaptado o suficiente ou até ter decorrido tempo suficiente
  retornar o melhor indivíduo em população, de acordo com FN-ADAPTA
-----------------------------------------------------------------------------
função REPRODUZ(x, y) retorna um indivíduo
  entradas: x, y, indivíduos pais
  n ← COMPRIMENTO(x)c ← número aleatório de 1 a n
  retornar CONCATENA(SUBCADEIA(x, 1 c), SUBCADEIA(y, c + 1, n))
```

Figura 27 – Um algoritmo genético. O algoritmo é igual ao que foi representado na figura 25, com uma variação: em sua versão mais popular, cada união de dois pais produz apenas um descendente, e não dois

Fonte: Russell e Norvig (2013, p. 109).

Resumo

Vimos nesta unidade os fundamentos da inteligência artificial (IA), área da ciência da computação que se concentra no desenvolvimento de algoritmos e sistemas que podem realizar tarefas que normalmente exigem inteligência humana. Isso inclui tarefas como reconhecimento de padrões, aprendizado de máquina, visão computacional, processamento de linguagem natural e tomada de decisões.

Apresentamos diversas áreas que contribuíram para a criação da IA e sua história, que começa na década de 1950, quando os cientistas passaram a explorar a possibilidade de criar máquinas que pudessem pensar e agir como seres humanos. Atualmente, a IA está em constante evolução, com avanços em diversas áreas. Trata-se de uma área emocionante e em constante evolução, com o potencial de transformar radicalmente muitos aspectos de nossas vidas.

Fundamentos importantes foram vistos na unidade, como o agente inteligente, que é um programa de computador que recebe entradas do ambiente ao seu redor e toma decisões para atingir um objetivo específico. O agente é capaz de perceber seu ambiente e interagir com ele, e pode ser programado para se comportar de maneiras diferentes para atingir um objetivo específico. O comportamento do agente pode ser orientado por regras ou ser baseado em aprendizado de máquina.

Vimos a forma de resolver problemas utilizando algoritmos de busca, que são métodos computacionais para encontrar soluções em um espaço de busca, e aprendemos que uma solução é um estado que atende a um conjunto de restrições ou critérios. Existem dois tipos de algoritmos de busca: busca sem informação e busca com informação.

A busca sem informação, também conhecida como busca cega, é um tipo de algoritmo de busca que não usa informações adicionais para ajudar na escolha de uma solução. Isso significa que o algoritmo de busca é completamente dependente do espaço de busca e de suas operações para encontrar uma solução. Exemplos de algoritmos de busca sem informação incluem busca em largura, busca em profundidade e busca de custo uniforme.

Já a busca com informação, também conhecida como busca heurística, é um tipo de algoritmo de busca que usa informações adicionais para ajudar a escolher uma solução. Essas informações adicionais são conhecidas como heurísticas e podem ser usadas para orientar a pesquisa em direções que têm

maior probabilidade de levar a soluções melhores. Exemplos de algoritmos de busca com informação incluem a busca A* e a busca de primeira escolha.

Em geral, os algoritmos de busca com informação tendem a ser mais eficientes e encontrar soluções mais rapidamente do que os algoritmos de busca sem informação, mas podem ser mais complexos e requererem mais recursos computacionais para implementação.

Exercícios

Questão 1. (FCC 2019, adaptada) Um analista necessita desenvolver uma aplicação chatbot que simule um ser humano na conversação com as pessoas. Para isso, ele deve usar a pesquisa em processamento de linguagem natural (PLN), que envolve três aspectos da comunicação, quais sejam:

A) Som, ligado à fonologia; estrutura, que consiste em análises morfológica e sintática; e significado, que consiste em análises semântica e pragmática.

B) Áudio, ligado à fonologia; estrutura, que consiste em análises de línguas estrangeiras; e significado, que consiste em análises semântica e pragmática.

C) Conversação, ligada à tecnologia de chatbot; semântica, que consiste em análises de línguas estrangeiras; e arquitetura spelling, que realiza análises sintática e pragmática.

D) Business intelligence, ligado à tecnologia Olap; mining, que consiste em análises de línguas em geral; e spelling, que realiza as funções de chatbot.

E) Áudio, ligado à fonologia; estrutura, que consiste em análises semântica e pragmática; e significado, que consiste em análises das línguas em geral.

Resposta correta: alternativa A.

Análise da questão

O processamento de linguagem natural (PLN) consiste no desenvolvimento de modelos computacionais para a execução de tarefas que dependem de informações expressas em alguma linguagem natural. Ele é usado para entender e gerar texto e fala em linguagem natural, sendo que esse processamento pode ser aplicado a chatbots. Os três pilares do PLN são os descritos a seguir.

- **Som**: ligado à fonologia, que é o estudo das regras que governam a combinação e a distribuição dos sons da língua.

- **Estrutura**: ligada à morfologia, que é o estudo das palavras e das suas estruturas, e à sintaxe, que é o estudo das regras que governam a ordem das palavras e a construção das frases.

- **Significado**: ligado à semântica, que é o estudo do significado das palavras, das frases e dos textos, e à pragmática, que é o estudo das regras que governam o uso da língua em contextos sociais e comunicativos.

Questão 2. No contexto de agentes inteligentes, Peas é um acrônimo que representa um modelo de quatro componentes que descreve a especificação de um agente inteligente. Os quatro componentes são: **p**erformance (desempenho), **e**nvironment (ambiente), **a**ctuators (atuadores) e **s**ensors (sensores). A respeito desses componentes, avalie as afirmativas:

I – Environment diz respeito ao ambiente em que o agente inteligente opera, podendo ser físico, virtual ou uma mistura de ambos.

II – Actuators diz respeito aos mecanismos que permitem que o agente compreenda o ambiente e seja capaz de receber informações.

III – Sensors diz respeito aos mecanismos que permitem que o agente realize ações no ambiente, como motores.

É correto o que se afirma em:

A) I, apenas.

B) III, apenas.

C) I e III, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa A.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: o termo **environment**, que significa ambiente, representa o ambiente externo com o qual o agente inteligente deve interagir. O ambiente pode ser físico, virtual ou uma mistura de ambos. Além disso, pode incluir objetos, outros agentes, obstáculos e informações.

II – Afirmativa incorreta.

Justificativa: o termo **actuators**, que significa atuadores e diz respeito ao conjunto de mecanismos que possibilitam que o agente realize ações no ambiente com o qual interage. Os atuadores podem ser dispositivos físicos, como braços robóticos, bem como softwares que controlam a interação. O texto da afirmativa diz respeito aos sensores, não aos atuadores.

III – Afirmativa incorreta.

Justificativa: o termo **sensors**, que significa sensores, diz respeito aos elementos que conferem "sentidos" aos agentes inteligentes. Por meio dos sensores, os agentes são capazes de receber informações a respeito do ambiente. Os sensores podem incluir câmeras, termômetros e microfones. O texto da afirmativa diz respeito aos atuadores, não aos sensores.