Técnicas de busca para inteligência artificial

Você vai compreender os conceitos e as aplicações de técnicas de busca para inteligência artificial, com abordagem das estruturas e das estratégias para a busca em espaços de estados, as buscas cega e heurística, além dos métodos de busca local e dos algoritmos genéticos.

Prof. Sérgio Assunção Monteiro

1. Itens iniciais

Propósito

Compreender os conceitos e as aplicações das técnicas de busca para inteligência artificial permitirá que o futuro profissional de tecnologia da informação seja capaz de selecionar os algoritmos mais adequados para a sua aplicação.

Objetivos

- Descrever as estruturas e as estratégias para a busca em espaços de estados.
- Descrever os aspectos relacionados à busca cega e à busca heurística.
- Caracterizar a busca local e os algoritmos genéticos.

Introdução

Temos a percepção de que muitas situações práticas do cotidiano da sociedade poderiam ser melhoradas, como:

- Os semáforos de trânsito poderiam se comportar de um modo diferente, dependendo do contexto.
- Uma alimentação econômica e composta pelos nutrientes adequados para manter nossa saúde em bom estado poderia ser mais acessível.
- Os meios de transporte público poderiam ser mais bem organizados, de modo a oferecer serviço de qualidade para a população em geral.
- As carteiras de investimento poderiam ser montadas de forma que nos trouxessem o melhor rendimento possível, com o menor risco.

A lista é gigantesca. Esse processo de melhoria é chamado de otimização. É uma área da matemática aplicada que usa recursos computacionais para tratar de problemas nas mais diversas áreas.

Os problemas de otimização têm algumas etapas bem distintas. Veja!

- Modelagem: é a formalização de um problema, descrita em uma linguagem com características bem específicas e que pode ser utilizada por programas.
- Resolução: é a aplicação de algoritmos para resolver os problemas que foram modelados.
- Análise dos resultados: consiste na crítica da solução e de sua aplicação. Pode levar à necessidade de voltar para a modelagem do problema para aperfeiçoamentos.

Os algoritmos tradicionais de otimização nem sempre são os mais adequados para a aplicação a determinados problemas. Assim, foram desenvolvidas diversas técnicas de inteligência artificial que tentam suprir as dificuldades desses algoritmos, por meio de processos iterativos e evolutivos na busca pela melhor solução para um determinado problema.

Vamos ver os conceitos relacionados a algumas das principais técnicas de inteligência artificial aplicadas à busca. Acompanhe!

Estruturas e estratégias para a busca em espaços de estados — contextualização

Otimização combinatória

Diversas situações podem ser formalizadas como formulações matemáticas. Muitos dos problemas enquadram-se na área de **otimização combinatória**.

Primeiramente, precisamos entender o que é otimização. Ela consiste em formulações matemáticas e métodos para resolver esses problemas. As formulações matemáticas são modelagens de situações, que podem ser descritas por meio de variáveis e regras, que relacionam essas variáveis com restrições e funções objetivo. Estas podem ser do tipo minimização ou maximização, quando tratamos de apenas uma função objetivo por formulação; ou, ainda, podem ter funções multiobjetivo, bem mais complexas de serem resolvidas.

Uma formulação geral de otimização com apenas uma função objetivo pode ser descrita como:

$$\begin{aligned} &\min f(x) \\ &\text{sujeito a: } g_i(x) \in D, \forall i = 1, \ldots, m \end{aligned}$$

$f(x)$ é a função objetivo que, nesse caso, queremos minimizar, e $g_i(x)$ corresponde às famílias de restrições que a solução do problema deve satisfazer. A área mais comum de aplicação de otimização é a programação linear, em que todas as variáveis e as funções envolvidas são lineares. Um problema de programação linear pode ser formulado como:

$$\begin{aligned} &\min c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\ &\text{sujeito a:} \\ &g_i(x) \leq b, \forall i \in I \\ &h_j(x) \geq d, \forall j \in J \\ &q_k(x) \leq h, \forall k \in K \\ &x_r \geq 0, \forall r = 1, \ldots, n \end{aligned}$$

Em que:

x é o vetor com n componentes, que representa a variável de decisão, e cada um de seus componentes é maior ou igual a zero.

A família de funções $g(x)$ associa-se às restrições do tipo menor ou igual.

A família de funções $h(x)$ relaciona-se às restrições do tipo maior ou igual.

A família de funções $q(x)$ relaciona-se às restrições do tipo menor ou igual.

Os vetores b, d e h correspondem aos termos independentes, e o vetor c corresponde aos coeficientes da função objetivo.

Para esclarecer, veja o exemplo.

$$\begin{gathered}\min 5x_1 + 10x_2 \\ \text{sujeito a:} \\ x_1 + 2x_2 \leq 20 \\ 4x_1 + x_2 \geq 10 \\ x_1 \geq 0 \text{ e } x_2 \geq 0\end{gathered}$$

As variáveis de decisão são x_1 e x_2. As restrições são:

$$x_1 + 2x_2 \leq 20 \text{ e } 4x_1 + x_2 \geq 10$$

Ambas as variáveis devem ser maiores ou iguais a zero.

Agora, precisamos resolver o problema. Para isso, vamos utilizar um método chamado **solução gráfica**. O primeiro passo consiste em traçar as funções do conjunto de restrições. Assim, obtemos o gráfico a seguir.

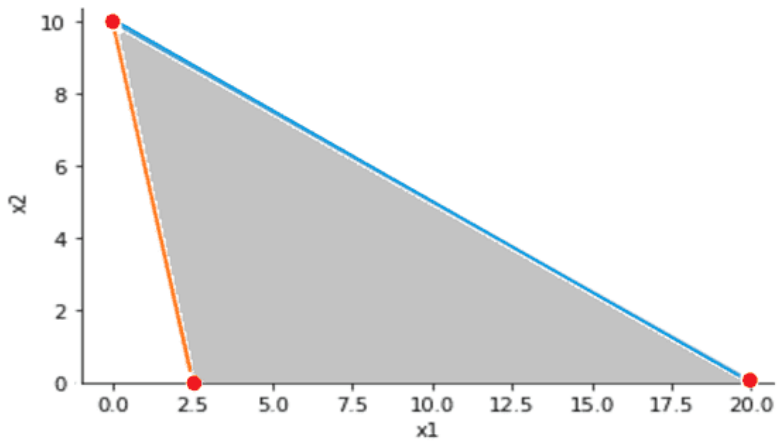


Gráfico: região viável do problema exemplo.

A região cinza do gráfico é chamada de região viável, o que significa que a solução do problema está nesse espaço. No caso da programação linear, a solução, quando existir, sempre estará em um dos vértices da região viável. No exemplo, precisamos analisar três soluções candidatas: (2.5;0), (20;0) e (0;10). Esses pontos correspondem aos vértices da região viável do gráfico anterior e estão destacados com um círculo vermelho. Agora, precisamos testar essas soluções candidatas na função objetivo da seguinte forma:

Para o vértice (2.5;0), temos que z = 5(2.5) + 10(0) = 12.5.

Para o vértice (20;0), temos que z = 5(20) + 10(0) = 100.

Para o vértice (0;10), temos que z = 5(0) + 10(10) = 100.

Como estamos trabalhando com um problema de minimização, então, a solução ótima é:

$$x_1 = 2.5 \text{ e } x_2 = 0$$

Dica

Apesar de ser bastante útil para resolver problemas com poucas variáveis, o método de solução gráfica é bastante limitado para resolver problemas com muitas variáveis. Para problemas de grande dimensão, utilizamos métodos mais sofisticados, como o Simplex.

Otimização discreta

A programação linear é bastante importante e tem muitas aplicações práticas, mas existem algumas áreas mais específicas da otimização que tratam de uma categoria de problemas muito interessante e com diversas aplicações na tomada de decisão que é a **otimização discreta**.

A formulação geral de um problema de otimização discreta é bastante parecida com a de um problema de programação linear, diferenciando-se, essencialmente, pelos valores que suas variáveis de decisão podem assumir, valores discretos. Estes pertencem a conjuntos enumeráveis, por exemplo, o conjunto dos números inteiros. A seguir, veja uma formulação para otimização discreta com apenas uma função objetivo.

$$\begin{aligned}
&\min c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
&\text{sujeito a:} \\
&\quad g_i(x) \leq b, \forall i \in I \\
&\quad h_j(x) \geq d, \forall j \in J \\
&\quad q_k(x) \leq h, \forall k \in K \\
&x_r \in M, \forall r = 1, \ldots, n
\end{aligned}$$

Atenção

A diferença dessa formulação para a de programação linear é que as variáveis de decisão pertencem a um conjunto discreto . Quando é o conjunto dos números inteiros, representado por , então, o problema é de otimização inteira. Quando o conjunto é composto apenas pelos elementos 0 e 1, o problema é de otimização combinatória.

A otimização discreta tem aplicações em diversos problemas práticos, como o de caminhos mínimos, o de rotas mínimas, o da mochila, o da alocação de recursos, entre muitos outros. Apesar de já existirem vários algoritmos para resolver problemas dessa área, ainda hoje é um grande desafio resolver muitos deles em um tempo considerado razoável. Isso ocorre devido à estrutura da formulação, além, é claro, da dimensão do problema em si. Portanto, é necessária uma abordagem alternativa para tratar desses casos. Uma que teve bastante sucesso e, ainda está em evolução, são as técnicas de inteligência artificial.

Sistemas de inteligência artificial

A inteligência artificial é uma área da ciência que estuda e descreve métodos que incorporam aspectos da inteligência observável. No artigo *What is artificial intellingence?* (O que é inteligência artificial?), McCarthy (2007) aborda, de uma forma muito interessante, aspectos da inteligência artificial. Resumidamente, o que temos são modelos, algoritmos e arquiteturas que são combinados para resolver problemas utilizando, para isso, programas de computadores.

Um sistema de inteligência artificial é composto por:

- Agentes
- Ambientes

Os agentes agem em um ambiente e um ambiente pode conter outros agentes.

> Um **agente** é qualquer coisa que tem a capacidade de perceber seu ambiente por meio de sensores e atuar nesse ambiente por meio de atuadores.

Agente humano

Possui órgãos sensoriais, ou seja, possui sensores, como olhos, ouvidos, nariz, língua e pele, e outros órgãos que são os atuadores, como mãos, pernas e boca.

Agente robótico

Tem como sensores as câmeras e outros dispositivos responsáveis por perceber a informação do ambiente e possui, como atuadores, os motores, por exemplo.

Agente de software

É acionado por meio de eventos e atua com o processamento da entrada dos dados.

Sobre o agente, existe uma terminologia que nos ajuda a entender melhor sua importância nesse contexto (Russell; Norvig, 2004), dada por:

Medida de desempenho do agente

É a forma de medir o grau de sucesso de um agente por meio de critérios bem estabelecidos. Por exemplo, qual o percentual de acerto de um agente para um determinado conjunto de testes.

Comportamento do agente

São as ações realizadas por um agente depois de uma determinada sequência de percepções. Por exemplo, a percepção aqui se refere às informações que o agente recebe ao longo de suas execuções. No caso de um problema de otimização, o agente recebe informações a respeito do número de iterações, como as soluções melhoraram ao longo do tempo.

Percepção

São as entradas percebidas pelos sensores do agente em uma determinada instância.

Sequência de percepção

Corresponde à história do que foi percebido por um agente.

Função do agente

É um mapa que relaciona uma ação à sequência de percepções.

Função de utilidade

É interessante perceber como os conceitos relacionados ao agente fazem analogias ao comportamento do ser humano. Até mesmo a incorporação de incerteza nos métodos faz parte desse processo.
Outro conceito que devemos destacar é o de **racionalidade**. A ideia é fornecer ao agente a capacidade de ter bom senso de julgamento. Portanto, ela está relacionada às ações e aos resultados esperados, dependendo do que o agente percebeu. Realizar ações com o objetivo de obter informações úteis é uma parte importante da racionalidade. Isso nos faz pensar sobre como deveria ser um agente ideal. Ora, ele é capaz de tomar as decisões mais adequadas, de modo que suas ações maximizem sua medida de desempenho, com base na sua sequência de percepção e de conhecimento.

A racionalidade de um agente depende dos seguintes aspectos:

- Medidas de desempenho, que determinam seu grau de sucesso.
- Sequência de percepção do agente.
- Conhecimento prévio do agente sobre o meio ambiente.
- Ações que o agente pode realizar.

Um agente racional sempre fará a escolha correta da ação que deve ser executada.

> A ação correta corresponde àquela que faz com que o agente seja mais bem-sucedido na sequência de percepção dada.

O problema que o agente resolve é caracterizado, portanto, por medida de desempenho, ambiente, atuadores e sensores.

Estrutura de agentes inteligentes

É a combinação entre arquitetura e programa agente. Quando falamos sobre arquitetura, estamos nos referindo à máquina ou ao meio no sentido de instrumento, em que um agente executa. Já o programa agente corresponde a uma implementação de uma função do agente. Existem algumas classificações a respeitos dos agentes. Veja.

Agentes de reflexo simples

Escolhem ações apenas com base na percepção atual. Eles são racionais apenas se uma decisão correta for tomada com base no preceito atual. Eles utilizam a regra de condição-ação que mapeia um estado (condição) para uma ação. Ou seja, dado que foi verificada determinada situação, qual deve ser a ação que o agente vai aplicar. Seu ambiente é completamente observável.

Agentes de reflexo baseados em modelo

Usam um modelo do mundo para escolher suas ações. Eles mantêm um estado interno, que é uma representação de aspectos não observados do estado atual, dependendo do histórico de percepção. Atualizar o estado requer informações sobre o modo que o mundo evolui, ou seja, como as ações do agente afetam o mundo. O termo modelo refere-se ao conhecimento sobre como acontecem as coisas no mundo.

Agentes baseados em objetivos

Escolhem suas ações a fim de atingir metas. Estas, ou ainda, objetivos, são a descrição de situações desejáveis, isto é, o que o agente pretende atingir. Essa abordagem é mais flexível do que o agente de reflexo simples, pois ele possui o modelo do conhecimento necessário para dar suporte a uma decisão, permitindo, assim, modificações.

Agentes baseados em utilidades

Escolhem ações com base em uma preferência (utilidade) para cada estado. Isso significa que a escolha das ações é feita por meio da verificação de qual delas leva para um estado melhor, de modo que o agente atinja seus objetivos com maior chance de sucesso.

Uma das situações que deve ser observada quando construímos um modelo é o estabelecimento das metas. Quando estas possuem objetivos conflitantes, precisamos associar medidas de incerteza a elas e dar um valor de retorno (recompensa) para o sucesso em relação à importância de uma meta.

A natureza dos ambientes

Como vimos até aqui, o ambiente em que o agente opera é uma parte fundamental na construção de um modelo de inteligência artificial. Por exemplo, alguns agentes operam em um ambiente inteiramente artificial restrito à entrada de dados via teclado, banco de dados, sistemas de arquivos de computador e saída em uma tela. Outros agentes, no entanto, podem atuar em sistemas mais complexos que estendem aspectos da realidade com o mundo digital.

Em especial, com a evolução da internet das coisas, ficou mais evidente como sistemas computacionais complexos podem interagir com dispositivos conectados à internet. A tecnologia permite que condições ambientais sejam monitoradas a distância e que dispositivos sejam programados de forma a fazer ajustes para manter o equilíbrio do meio que está sob monitoramento.

Exemplo

É possível realizar determinados procedimentos médicos à distância em locais com escassez de especialistas. Também é possível monitorar condições de desmatamento e queimadas de florestas e, muitas vezes, atuar de modo a minimizar danos.

Teste de Turing.

Sob o ponto de vista histórico, o ambiente artificial mais famoso é o ambiente de **Teste de Turing**. Nele, um agente real e outro artificial são testados sob condições iguais. Esse é um ambiente muito desafiador, pois é difícil para um agente de software funcionar tão bem quanto um ser humano. Esse teste é usado para medir o sucesso de um comportamento inteligente de um sistema.

A ideia é que duas pessoas e um agente artificial participem do teste. Das duas pessoas, uma vai desempenhar o papel de testador, ou seja, vai ser submetida ao teste. Cada uma das pessoas fica em quartos diferentes. O testador não sabe quem é o agente artificial (a máquina) e quem é o ser humano. Então, o testador faz as perguntas digitando e enviando-as para ambas as inteligências: a outra pessoa e o agente artificial. Agora, ou a pessoa, ou o agente vai responder à pergunta e enviar a resposta para o testador. Se o testador não for capaz de distinguir se a resposta foi do agente artificial ou se foi uma resposta humana, então, o agente é considerado inteligente. Esse teste visa enganar o testador.

Quando estudamos o **ambiente** no qual o agente está, devemos analisar suas propriedades, como vamos ver a seguir:

Discreto ou contínuo

Ocorre quando existe um número limitado de estados distintos e claramente definidos do ambiente, como vimos quando falamos sobre a otimização discreta. Uma situação em que isso ocorre, por exemplo, é no jogo de xadrez, em que as possibilidades do jogo são bem definidas apesar de serem gigantescas. No caso de domínio contínuo, as soluções das variáveis podem assumir valores reais. Um exemplo de aplicação de domínio contínuo é o de um robô se deslocando em ambiente tridimensional.

Observável ou parcialmente observável

Se for possível determinar o estado completo do ambiente em cada ponto do tempo a partir das percepções, ele é observável; caso contrário, é apenas parcialmente observável.

Estático ou dinâmico

Se o ambiente não muda enquanto um agente está agindo, então, ele é estático; caso contrário, é dinâmico.

Com presença ou não de outros agentes

Pode haver outros agentes no ambiente, os quais podem ser do mesmo tipo ou de tipo diferente do agente.

Acessível ou não acessível ao agente

Se o agente pode ter acesso ao estado completo do ambiente, então, o ambiente é acessível a esse agente.

Determinístico e não determinístico

Se o próximo estado do ambiente for completamente determinado pelo estado atual e pelas ações do agente, então, o ambiente é determinístico; caso contrário, é não determinístico.

Ambiente episódico ou sequencial

Cada episódio consiste na capacidade de percepção do agente e, em seguida, em sua atuação. A qualidade de sua ação depende apenas do episódio em si. Os episódios subsequentes não dependem das ações dos episódios anteriores. Ambientes episódicos são muito mais simples porque o agente não precisa pensar no futuro.
Outra situação, e bem mais complexa, ocorre quando o ambiente é sequencial. Isso acontece quando a execução de uma ação pode afetar as decisões futuras. Por exemplo, no jogo de xadrez, a escolha de uma jogada pode impactar as demais escolhas.

Terminologia dos algoritmos de busca

Os algoritmos de busca são métodos de inteligência artificial usados para resolver problemas. Eles são aplicados por meio dos agentes racionais baseados em objetivos. Nesse contexto, existe uma terminologia usada para nos referir aos diversos elementos envolvidos. Confira, a seguir, os principais termos da terminologia dos algoritmos de busca.

Principais termos da terminologia dos algoritmos de busca				
Estado Corresponde a uma configuração. Ao longo da execução do agente, ele verifica se um determinado estado corresponde à solução que ele tem como objetivo.	**Operador** Faz a transformação de um estado em outro.	**Espaço de busca** Representa um conjunto de soluções possíveis que um sistema pode ter.	**Estado inicial** É o estado em que a pesquisa começa.	**Função objetivo** É uma função que analisa o estado atual e retorna se é ou não o estado objetivo.
Problema de busca É composto pelo espaço de busca, estado inicial e função objetivo. Também é chamado de instância do problema.	**Grafo do espaço do problema** Representa os estados do problema. Os estados são mostrados por nós e os operadores, por arestas.	**Profundidade de um problema** Corresponde à sequência de operadores do estado inicial ao estado objetivo.	**Complexidade do espaço** Refere-se à quantidade máxima de memória necessária para encontrar uma solução.	**Completude** É a propriedade que garante que o algoritmo vai encontrar uma solução, desde que ela exista. Ou seja, se o agente sempre acha a solução.
Admissibilidade É a propriedade de um algoritmo encontrar uma solução ótima de forma otimizada.	**Fator de ramificação** É o número médio de nós filhos no grafo do espaço do problema.	**Profundidade** É o comprimento do caminho mais curto do estado inicial ao estado objetivo.	**Ações** Descreve todas as ações disponíveis ao agente que terão efeito nos estados do sistema.	**Modelo de transição** É uma descrição do que cada ação faz.

Principais termos da terminologia dos algoritmos de busca				
Modelo de transição É uma função que atribui um custo numérico a cada caminho escolhido ao longo da execução do algoritmo de busca.	**Custo do caminho** É uma função que atribui um custo numérico a cada caminho escolhido ao longo da execução do algoritmo de busca.	**Solução** É uma sequência de ações que vai do nó inicial ao nó objetivo, não necessariamente ótimo.	**Solução ótima** Corresponde à solução com menor custo entre todas as soluções.	**Otimalidade** Se uma solução encontrada para um algoritmo é garantidamente a melhor solução (custo do caminho mais baixo) entre todas as outras soluções, então, tal solução é considerada uma solução ótima.

Espaço de estados

Os algoritmos de busca no espaço de estados são usados com bastante frequência em inteligência artificial para resolver problemas. Isso ocorre pela facilidade de desenvolver técnicas que reproduzem comportamento com características de inteligência que incluem, entre outras características, a adaptabilidade, que é a capacidade de um agente fazer uso de estratégias durante a execução do algoritmo que aumentem a chance de atingir o objetivo com o menor esforço possível. Nesse processo, precisamos modelar um agente que seja capaz de realizar as ações mais adequadas para atingir a solução objetivo. Ao executar uma ação, o agente modifica o estado corrente do ambiente em que está contextualizado.

Lembre-se de que a instância de um problema é definida por:

- Estado inicial, que representa a configuração do ambiente no início da execução do agente.
- Conjunto de ações que o agente pode executar.
- Descrição do objetivo que o agente deseja obter.

Ao longo das iterações, o agente realiza uma sequência de ações que implica a mudança da configuração do estado até obter a solução ótima do problema ou, pelo menos, obter uma solução aceitável.

Atenção

Aceitável significa que a solução obtida não é necessariamente a ótima, mas satisfaz determinados critérios preestabelecidos.

Para definir um espaço de estados, precisamos de um conjunto de estados — que vamos representar pela letra **S** — e por um conjunto de ações que mapeiam um estado em outro — que representamos pela letra **A**. Para entendermos melhor como isso funciona, vamos considerar um exemplo: o mundo de um robô aplicado na irrigação de plantas.

Nesse mundo, o agente é um robô com a função de regar duas plantas. Nesse ambiente, o robô se desloca em um trilho para chegar até elas.

Além disso, existe uma posição de repouso para a qual o robô vai voltar sempre que terminar de irrigar uma planta. Ainda para simplificar um pouco mais o problema, quando o robô estiver na posição de repouso, ele vai ser abastecido de água.

Cada vaso de planta possui um sensor que informa se ela precisa ser regada. Portanto, as plantas podem precisar ou não de água, e o robô pode realizar as seguintes ações:

ir_para_planta1 ir_para_planta2 voltar_da_planta1_para_repouso voltar_da_planta2_para_repouso regar_planta1 regar_planta2

Representação de estados

Os estados são as configurações que podem existir em um ambiente em que o agente atua. Por meio de estruturas, podemos representá-los. Cada componente da estrutura representa um atributo da configuração de um estado. Por exemplo, no caso do robô aplicado à irrigação de plantas, cada estado pode ser representado por uma estrutura da forma [X, P1, P2], em que:

X

É o atributo que indica a posição que o robô está. Ele pode ser um dos seguintes elementos:

- 0, se o robô estiver na posição de repouso.
- 1, se o robô estiver na posição da planta 1.
- 2, se o robô estiver na posição da planta 2.

P1

É o atributo relacionado ao sensor da planta 1. Ele pode ser um dos seguintes elementos:

- 1, se o sensor da planta 1 foi acionado e, portanto, indica que a planta 1 precisa de água.
- 0, indica que o sensor não foi acionado e, portanto, a planta 1 não precisa de água.

P2

É o atributo relacionado ao sensor da planta 2. É semelhante ao P1. Ele pode ser um dos seguintes elementos:

- 1, se o sensor da planta 2 foi acionado e, portanto, indica que a planta 2 precisa de água.
- 0, indica que o sensor não foi acionado e, portanto, a planta 2 não precisa de água.

Portanto, temos:

$$X \in \{0, 1, 2\}$$

$$P1 \in \{0, 1\}$$

$$P2 \in \{0, 1\}$$

Para exemplificar, se o robô estiver na posição da planta 1, o sensor da planta 1 estiver ativo e o sensor da planta 2 estiver desativado, então, o estado do robô é dado por [1, 1, 0].

O conjunto de estados para o robô aplicado à irrigação das plantas é:

$$S = \{[0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1], [2,0,0], [2,0,1], [2,1,0], [2,1,1]\}$$

Como podemos ver, o conjunto S, ou seja, o espaço de estados, possui 12 estados. Poderíamos ter chegado a esse mesmo resultado sem precisar enumerar o conjunto, por meio do princípio multiplicativo, da seguinte forma:

- Cada estado é representado pela estrutura: [X, P1, P2].
- O conjunto X possui 3 elementos que representamos pela notação |X|=3.
- Os conjuntos P1 e P2 possuem, cada um, dois elementos que representamos por |P1|=2 e |P2|=2.
- Portanto, a quantidade de elementos do conjunto S é dada por: |S| = |X|•|P1|•|P2|= 3 • 2 • 2 = 12

Representação das ações

Ainda no exemplo do robô aplicado à irrigação de plantas, vamos tratar das representações das ações. Quando o robô está em um determinado estado e uma ação é aplicada, então uma das duas situações ocorre: ou ele permanece no mesmo estado, ou vai para outro estado. Para representar a execução de uma ação, vamos utilizar a notação:

> Operador(ação, estado_atual, próximo_estado).

Os comportamentos esperados para o robô envolvem os procedimentos descritos a seguir.

Se a planta 1 precisa ser irrigada, então, temos:

Se P1 = 1, então: Operador(ir_para_planta1, [0,1, P2], [1, 1, P2]) Operador(regar_planta1, $[1,1,P2], [1,0,P2]$) Operador(voltar_da_planta1_para_repouso, $[1,0,P2], [0,0,P2]$)

Se a planta 2 precisa ser irrigada, então, temos:

Se P2 = 1, então: Operador(ir_para_planta $2, [0,P1,1], [2,P1,1]$) Operador(regar_planta $2, [2,P1,1], [2,P1,0]$) Operador(voltar_da planta 2- para_repouso, $[2,P1,0], [0,P1,0]$)

O robô sempre começa da posição de repouso. Se um dos sensores estiver ativo, o robô vai se deslocar até a respectiva planta. Em seguida, vai regá-la, desse modo, o sensor será desativado. Por fim, o robô volta para a posição de repouso.

O conjunto de ações para o exemplo do robô aplicado à irrigação de plantas é dado por:

$$A = \{Operador(ir\_para\_planta\_1, [0,1,P2], [1,1,P2]), Operador(regar\_planta1, [1,1,P2], [1,0,P2]), Operador(voltar\_da\_planta\_1\_para\_repouso, [1,0,P2], [0,0,P2]), Operador(ir\_para\_planta\_2, [0,P1,1], [1,P1,1]), Operador(regar\_planta2, [2,P1,1], [2,P1,0]), Operador(voltar\_da\_planta\_1\_para\_repouso, [2,P1,0], [0,P1,0])\}.$$

Estratégias de busca

As estratégias dos algoritmos de busca determinam a ordem em que os nós são explorados. Existem três estratégias de busca principais. Conheça-as!

Amplitude primeiro

Esses métodos são mais conhecidos como métodos de busca em largura. Começando do nó inicial, a estratégia de largura primeiro expande todos os nós vizinhos. Em seguida, para cada um dos nós, repetimos o mesmo processo de explorar os nós vizinhos. Essa sequência de exploração é repetida até que o algoritmo encontre a solução objetivo. Esses algoritmos são conhecidos em inglês como *breadth-first search* e, normalmente, são referenciados pela sigla BFS.

Melhor primeiro

Essa estratégia explora os nós do grafo de estados por meio da expansão do nó mais promissor, ou seja, do nó que, aparentemente, tem maior chance de estar no caminho da solução ótima. Quando o mérito — medida do valor da solução — dos nós é feito por uma função de custo, essa estratégia sempre expande os nós em ordem não decrescente de seus custos. Os algoritmos que utilizam essa estratégia são conhecidos em inglês como *best-first*.

Profundidade primeiro

Os métodos de busca em profundidade exploram repetidamente o nó gerado mais recentemente. Esses algoritmos são conhecidos em inglês como *depth-first search* e, normalmente, são referenciados pela sigla DFS.

Cada uma das estratégias produz algoritmos de busca diferentes. Existem várias combinações e extensões das estratégias *best-first* e *depth-first*.

A escolha da estratégia dos algoritmos de busca também depende do problema a ser resolvido na prática. O caminho natural que devemos seguir é, sempre que possível, procurar pelas estratégias que já têm suas eficiências comprovadas por meio de testes e de uso em situações práticas.

Comentário

De modo geral, a estratégia de amplitude é a menos eficiente, pois, como ela expande todos os nós em um nível mais superficial antes que um nó em um nível mais profundo possa ser examinado, aumentam as chances de serem feitos testes desnecessários e, assim, prejudicar o desempenho do algoritmo.

Neste vídeo, o especialista fala sobre os conceitos de otimização e quando e como devem ser empregadas as estruturas e as estratégias para busca em espaços de estados. Assista para entender melhor!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Os métodos de busca no espaço de estados fazem parte da inteligência artificial e são muito utilizados na prática. Isso ocorre porque muitos problemas podem ter suas configurações e ações discretizadas. Assim, selecione a opção correta sobre a solução de um problema no espaço de estados.

A

Corresponde a uma sequência de escolha de ações.

B

Refere-se à solução ótima do problema.

C

É qualquer estado que pertença ao conjunto de estados do problema.

D

Corresponde a um estado que satisfaça algumas condições preestabelecidas de otimalidade.

E

É uma composição entre um determinado estado e a ação que levou até ele.

A alternativa A está correta.

Quando o agente inicia sua execução, ele parte de um estado inicial e, por meio de uma sequência de escolhas, transforma o estado inicial no estado objetivo, que é o estado que resolve o problema. O espaço de estados trata do mapeamento entre as ações e os estados, e não apenas de um caso específico. Além disso, não há uma garantia de que o estado sucessor conduzirá a uma solução ótima.

Questão 2

Os métodos de busca no espaço de estados podem ser utilizados para modelar situações tanto teóricas como práticas que têm em comum o fato de serem discretizáveis. O termo discretizável está associado à possibilidade de enumerar quais as possíveis configurações que um agente pode assumir ao longo da resolução de um problema. Assim, selecione a opção correta sobre os métodos de busca no espaço de estados.

A

São considerados métodos muito eficientes por sempre garantirem a obtenção da solução ótima de um problema.

B

O espaço de busca é constituído pelos estados e pelas ações que compõem o ambiente em que o agente está contextualizado.

C

São caracterizados pela velocidade com que chegam à solução ótima, podendo, assim, ser aplicados para resolver problemas reais.

D

São aplicados apenas para resolver problemas teóricos, pois, em situações reais, existem muitas situações que não podem ser modeladas.

E

Não podem ser aplicados para programar robôs, pois tratam apenas de instruções que se aproximam do linguajar humano.

A alternativa B está correta.

A eficiência de um método depende do modo como ele explora o espaço de buscas. O espaço de busca representa as configurações que um sistema pode assumir quando são aplicadas ações. A escolha das ações é feita quando alguma condição é satisfeita. Por exemplo, quando um sensor de temperatura é ativado, o agente pode ser acionado para tomar uma ação de modo a manter as condições do ambiente sob controle. Não há uma garantia de que o estado sucessor conduzirá a uma solução ótima. Os métodos podem ser teóricos ou reais e podem ser utilizados para a robótica, inclusive.

Busca cega e busca heurística — contextualização

A inteligência artificial é a ciência e a engenharia que estudam métodos de como fazer com que algoritmos reproduzam comportamentos inteligentes, ou seja, que sejam capazes de aprender com a experiência adquirida. O elemento da inteligência artificial que atua ativamente para colocar esse comportamento na prática são os agentes. A suposição básica é que esses agentes agem racionalmente, o que significa dizer que fazem as melhores escolhas para atingir uma solução alvo.

> O que é a solução alvo?

É o objetivo — ou a meta — que o agente pretende alcançar.

Exemplo

Em uma situação em que o agente é um robô aspirador de sujeira, e ele está em um ambiente com algumas salas equipadas com sensores que informam a necessidade de serem limpas, a solução alvo do robô é manter as salas sem sujeira.

Percebemos que o robô precisa ser capaz de se locomover para chegar até as salas e aspirá-las. Todo esse sequenciamento de ações pode ser mapeado em um espaço de estados. A partir da existência de mapas, então, é possível descrever um método de busca em um espaço de estados.

Um problema de busca consiste em três partes. Confira!

Espaço de estados

É o conjunto de todos os estados possíveis em que um agente pode estar.

Estado inicial

É o estado em que a busca começa.

Teste de objetivo

É uma função que verifica o estado atual e informa se ele é ou não o estado objetivo.

O sequenciamento das ações para atingir o objetivo de um problema é chamado de **solução para um problema de busca**. Trata-se de uma sequência de execuções de ações que transformam o estado inicial em estado objetivo. Esse sequenciado é realizado por meio de algoritmos de busca.

Tipos de algoritmos de busca fundamentais

Podem ser divididos em duas categorias. Conheça-as!

Busca cega

Também conhecida como busca sem informações. Em inglês, é chamada de *blind search* e, também, de *uninformed search*. Esse tipo de algoritmo funciona sem nenhuma informação sobre o espaço de busca com a única exceção para distinguir o estado objetivo — ou estado alvo, ou, ainda, estado meta — dos demais.

Busca heurística

Também conhecida como busca com informações ou busca direcionada (ou dirigida). Tenta reduzir a quantidade de buscas que devem ser feitas mediante escolhas inteligentes para os nós que são selecionados para serem expandidos. Ou seja, os nós com maiores chances de produzir uma boa solução são priorizados em relação aos demais. Para isso, esses algoritmos utilizam uma forma de avaliar a probabilidade de um determinado nó estar no caminho da melhor solução. Isso é feito com o uso de uma função heurística.

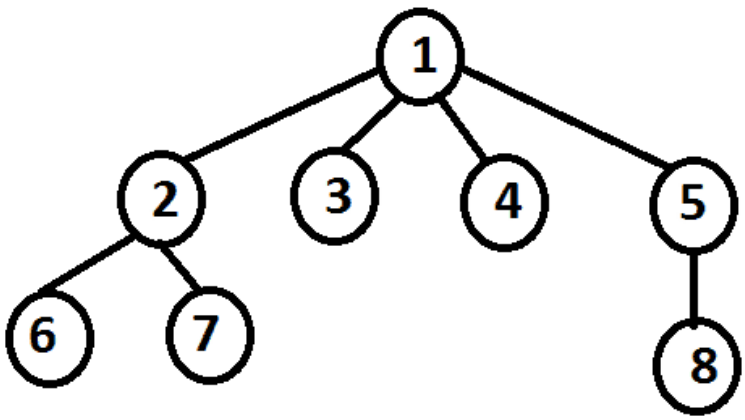
A seguir, vamos explicar com mais detalhes como os tipos de algoritmos funcionam.

Algoritmos de busca cega

Não têm informações adicionais sobre o estado alvo, exceto as que já fazem parte da definição do próprio problema. Outro ponto que devemos observar sobre esses algoritmos é que eles não consideram a qualidade de uma determinada configuração para obter a solução do problema. Conheça os principais algoritmos de busca cega:

Busca em largura (ou amplitude)

A busca passa pela árvore nível por nível, visitando inicialmente todos os nós no nível superior, depois todos os nós no segundo nível, e assim por diante. A seguir, veja um exemplo de como a busca em largura funciona.



Exemplo de execução da busca em largura.

A estratégia de busca em largura tem a vantagem de ser completa (se houver uma solução, ela será encontrada) e, além disso, será ótima. Para isso, o algoritmo de busca mantém todos os nós folhas — que são os nós da árvore de busca que não têm filhos — na memória. Essa característica é uma desvantagem do método, pois requer uma quantidade proibitiva de memória para resolver problemas em que existem muitas possibilidades a serem exploradas.

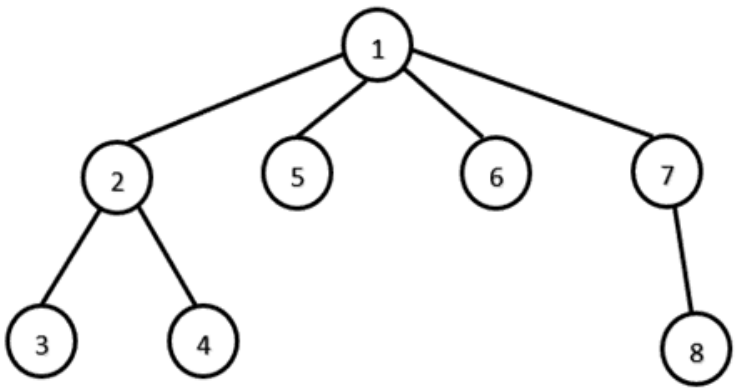
O custo computacional de tempo, que é definido como a complexidade de tempo da busca em largura é $O\left(b^d\right)$, em que:

- b é o fator de ramificação, que é a quantidade de filhos que um nó pode ter.
- d é a profundidade da solução, que é a distância do nó inicial até o nó alvo.

Além disso, a complexidade de espaço também é $O\left(b^d\right)$.

Busca em profundidade

Esse método explora todos os ramos da árvore, descendo até os nós folhas antes de explorar o próximo ramo. A seguir, veja um exemplo da forma como a busca em profundidade funciona.



Exemplo de execução da busca em profundidade.

A estratégia de busca em profundidade demanda menos memória do que a busca em largura, pois ela precisa armazenar apenas um único caminho da raiz da árvore até o nó folha. A complexidade de tempo da busca em profundidade é $O\left(b^m\right)$, em que m é a profundidade máxima da árvore e a complexidade de espaço é $O(b \cdot m)$.

Busca limitada pela profundidade

É uma variação da busca em profundidade. O processo de busca é limitado a um limite de profundidade especificado. Quando o agente atinge um nó em determinada profundidade, ele para de descer naquele ramo e passa para o próximo. Isso evita o problema potencial da busca em profundidade de descer um ramo indefinidamente. No entanto, a pesquisa com profundidade limitada é incompleta — se houver uma solução, mas apenas em um nível mais profundo do que o limite, ela não será encontrada. A complexidade de tempo da busca em profundidade é $O\left(b^l\right)$, em que b é o fator de ramificação — da mesma forma que nos métodos anteriores — e l é o limite de profundidade imposto ao algoritmo. Sua complexidade de espaço é $O(b \cdot l)$.

Busca em profundidade iterativa

Esse método faz buscas repetidas com limitação de profundidade. Ele combina características das buscas em profundidade e largura com a necessidade de utilizar menos memória que a busca em largura, semelhante à busca em profundidade, mas explorar todo o espaço de estados, semelhante à busca em largura. A ideia do algoritmo é começar com um limite de profundidade igual a 1 e explorar as soluções vizinhas. Caso não tenha encontrado a solução alvo, a profundidade é incrementada e, agora, o método passa a explorar uma profundidade igual a 2 e as soluções vizinhas. Esse processo segue até que a solução alvo seja obtida. A complexidade de tempo da pesquisa de aprofundamento iterativa é $O\left(b^d\right)$ e a complexidade do espaço é $O(b \cdot d)$.

Busca de custo uniforme

É uma variação da busca em largura. Ela associa um custo aos caminhos que partem do nó inicial. No processo de busca, em vez de se comportar como a busca em largura que expande o nó no nível mais superficial (raso), ela expande o nó que está no caminho com menor custo. Ou seja, percorrer diferentes

ramos pode ter custos diferentes. O objetivo é encontrar um caminho em que a soma cumulativa dos custos seja mínima.

Busca bidirecional

A ideia desse método é atuar nas duas direções: do nó inicial para o nó alvo e do nó alvo para o nó inicial. O processo de busca continua a ser de exploração e é necessário realizar diversos testes. Quando as buscas se encontram, o processo é concluído.

Heurísticas

As estratégias de busca heurísticas usam conhecimento específico do problema para encontrar a solução mais rapidamente. Por isso, também são chamadas de buscas informadas e, ainda, de buscas direcionadas. Os algoritmos utilizam uma função heurística para avaliar as chances de um determinado nó estar no caminho da solução. A ideia básica é a de avaliar um subconjunto de estados por meio da função heurística e expandir os que possuem melhores chances de melhorar o desempenho médio da busca. Essas funções são específicas para domínios diferentes de problemas. Um exemplo de uma função heurística é a distância euclidiana que, basicamente, calcula a distância entre dois pontos em um espaço euclidiano.

Conheça principais algoritmos de busca heurística:

Busca pelo melhor primeiro (best first search)

A ideia dessa estratégia de busca é bem simples. A cada iteração, o algoritmo faz uma avaliação dos nós por meio da função heurística e explora o que tiver a melhor nota de avaliação. Os nós são colocados em uma fila de prioridades e, em seguida, são explorados.

Greedy search

Essa estratégia de busca é traduzida para o português como busca míope ou busca gulosa, sendo esta última a forma mais comum. Ela sempre faz a escolha que parece ser a melhor naquele momento, portanto, trabalha na expectativa de que a escolha local leve a uma solução ótima global.

Busca A*

Essa estratégia é chamada de busca A-estrela. Ela é uma variação da *best first search*. Ela utiliza duas funções para estimar o custo da solução ótima. Sua função heurística é dada por:

$$< br > f(n) = g(n) + h(n) < br >$$

- A função g(n) obtém o custo do nó inicial até o nó n.
- A função h(n) obtém o custo estimado do caminho de menor custo de n até o nó objetivo.

IDA*

O nome IDA* é uma sigla para *iterative deepening A**, traduzido para o português como algoritmo A* iterativo em profundidade. É uma variação da estratégia A*, que tem um impacto significativo na redução do uso de memória. Basicamente, a IDA* aplica a estratégia A* em uma profundidade diferente, até que obtenha a solução do problema.

Neste vídeo, o especialista fala sobre os conceitos de busca cega e busca heurística, seus algoritmos e a aplicabilidade de cada um. Assista para entender melhor!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Os métodos de busca no espaço de estados fazem parte da inteligência artificial e podem ser aplicados em muitas situações práticas. Os problemas que esses algoritmos tratam são constituídos por partes fundamentais que serão exploradas para obter a solução alvo. Considere a seguinte situação: Um robô tem a capacidade de realizar uma única ação: "regar uma planta". Ele colocará água na planta sempre que o sensor que está instalado nela ficar ativado. Levando isso em consideração, selecione a opção correta.

A

Não é possível determinar o espaço de busca do problema, pois faltam informações.

B

O conjunto de estados alvo do exemplo é formado por apenas um estado alvo: "manter a planta hidratada".

C

O robô é o agente e seu conjunto de ações é composto pelas ações "mover-se até a planta", "regar a planta" e "encher o repositório do robô com água".

D

Trata-se de um exemplo de busca cega no espaço de estados.

E

Não está claro no exemplo quais são as ações do robô.

A alternativa B está correta.

O espaço de estados é composto pelos conjuntos de ações e de estados. Os estados do exemplo são dois: ou a planta está hidratada, ou não está hidratada. O conjunto de ações desse caso é: "regar a planta".

Questão 2

Os métodos de busca no espaço de estados executam uma sequência de passos para obter a solução alvo. Apesar de esse processo não ser determinístico, os métodos podem ser constituídos de estratégias para aumentar as chances de sucesso. Assim, selecione a opção correta a respeito dos métodos e dos problemas no espaço de busca.

A

Por meio do teste de objetivo, é possível determinar se o agente já obteve a sua meta.

B

Como o conjunto de estados é limitado, a quantidade de passos até atingir a solução alvo também é limitada.

C

O agente racional sempre faz escolhas que aumentem as chances de atingir a solução alvo de modo otimizado sem considerar o histórico de suas escolhas.

D

O espaço de estados é constituído por todas as configurações de estados que um agente pode assumir, no entanto, as ações são tratadas à parte, pois são representadas por outro conjunto.

E

O agente trabalha para obter a ação ótima, pois ela que determinará a solução ótima.

A alternativa A está correta.

Ao longo de suas execuções, um agente faz a transição entre os estados, mas suas escolhas visam atingir a solução alvo. Portanto, é necessário fazer essa verificação sempre que o agente muda de estado. O responsável explícito por isso é o teste de objetivo, que informa se o estado atual é ou não o estado objetivo.

Busca local e algoritmos genéticos — contextualização

A otimização combinatória tem aplicações em muitas áreas, como a ciência da computação, as engenharias de um modo geral e a administração, por exemplo. Os problemas de otimização combinatória são formados por:

- Variáveis de decisão que explicitam o que “deve ser feito”.
- Um conjunto de restrições que impõe limites sobre o que “pode ser feito” em relação às variáveis.
- Um domínio — conjunto que contém os valores que as variáveis de decisão podem assumir. No caso da otimização combinatória, esse conjunto é formado pelos elementos 0 e 1.
- Uma ou mais funções objetivo que podem ser de minimização ou de maximização.

O conjunto de soluções para um problema é a combinação dos valores que as variáveis de decisão podem assumir no conjunto domínio e que satisfazem as restrições do problema.

Função ótima global

Se a solução encontrada for a melhor possível, ou seja, se for a solução que **minimiza** (se o problema for de minimização) ou **maximiza** (se o problema for de maximização) a função objetivo, então, ela é chamada de função ótima global.

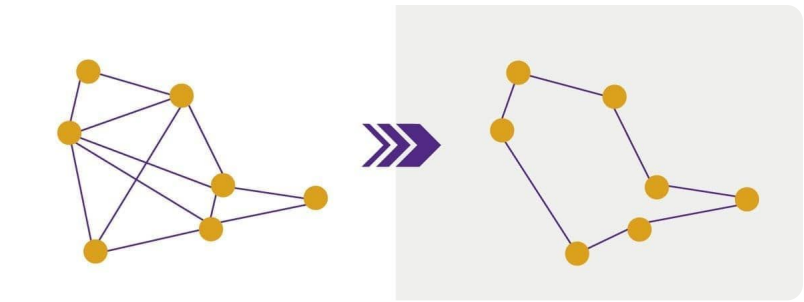
Função ótima local

Se a função é a melhor possível para um subconjunto das soluções do problema, então, ela é chamada de função ótima local.

Traveling salesman problem (TSP)

Um dos problemas de otimização combinatória mais bem estudados é o do caixeiro viajante — *traveling salesman problem* (TSP), em inglês —, que pode ser descrito da seguinte maneira: considere um vendedor (caixeiro viajante) que deseja visitar cada cidade de um conjunto de n cidades exatamente uma vez e terminar na mesma cidade na qual começou, em que a distância entre duas cidades quaisquer é conhecida.

> **Então, o problema que desejamos otimizar é: em que ordem o vendedor deve visitar as cidades, de modo que a distância percorrida seja mínima?**



Traveling salesman problem.

O TSP é um exemplo de problema de otimização combinatória, pois a solução dele consiste em escolher uma sequência de cidades de tal forma que a distância percorrida seja mínima e que, além disso, ele retorne para a cidade de origem. Embora seja um problema fácil de entender e tenha muitas aplicações práticas, sendo que as mais simples de visualizar são as de determinação de rotas e de cadeias logísticas, ele é um problema muito difícil de ser resolvido. Isso ocorre devido às diversas possibilidades que precisam ser verificadas.

Na prática, o que acontece é a aplicação de algoritmos heurísticos e, em alguns casos, de algoritmos aproximativos que não garantem a solução ótima global, mas produzem soluções aceitáveis, ou seja, que atendem a algumas condições que, se forem satisfeitas, tornam a solução útil para ser aplicada.

Vamos falar sobre duas famílias de algoritmos muito importantes, pois são aplicadas a diversos problemas de otimização combinatória com resultados muito bons na prática, que são os algoritmos de busca local e os algoritmos genéticos.

Busca local

É um método de aproximação bastante aplicado em otimização combinatória. Apesar de não garantir a solução ótima global, essa técnica garante que a solução encontrada é a melhor possível para um subconjunto de soluções. É essa característica que dá o nome à técnica.

> A melhor solução local, se existir, é chamada de **solução ótima local**.

Veja os prós e os contras da aplicação dessa técnica:

Vantagem	Desvantagem
Reduz o espaço das soluções para um subconjunto. Às vezes, essa redução pode levar para formulações bem conhecidas e com propriedades que podem ser exploradas, de modo que a obtenção de uma solução ótima local é menos custosa do que resolver o problema original.	Se a busca local encontrar a solução ótima local, pode ser que não seja a solução ótima global. Além disso, a redução do espaço de busca pode impor restrições ao problema que eliminem soluções locais.

Dica

Apesar das desvantagens, na prática, essa técnica funciona muito bem na solução de problemas de otimização combinatória grandes e complexos.

O espaço de busca que um algoritmo de busca local explora é definido por uma vizinhança de soluções viáveis. Ou seja, serão essas soluções que o algoritmo vai considerar para otimizar o problema. Nesse momento, devemos ter clareza de que o problema já foi modificado, ou seja, reduzimos suas possíveis soluções. O ideal é que tivéssemos reduzido o espaço de busca com a eliminação de soluções inúteis e, ainda assim, tivéssemos mantido a solução ótima global do problema, mas essa, infelizmente, é uma situação rara.

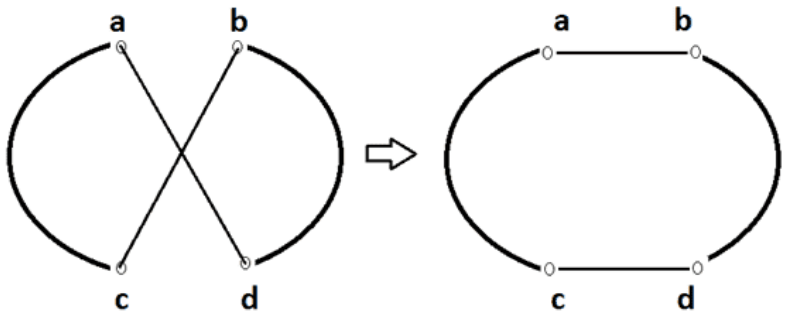
Ainda sobre as vizinhas de uma solução, dois estados são conectados se houver um operador que muda de um estado para outro por uma perturbação local. Essa é uma das características dos métodos de busca local que faz, inclusive, que sejam conhecidos como métodos de busca de vizinhança. A ideia desses operadores é que façam perturbações locais — devemos entender perturbação por modificação — para transformar um estado em outro. Esses operadores são desenvolvidos para tratar características específicas de uma categoria de problemas, de modo a explorar sua estrutura de vizinhança.

> O desenvolvimento de operadores eficazes que explorem estruturas da vizinhança de um problema é o principal desafio para os métodos de busca local.

Um dos exemplos mais simples de um operador de perturbação local é o **2-opt** para o problema do **caixeiro viajante** simétrico, mostrado a seguir.

caixeiro viajante

O problema do caixeiro viajante pode ser simétrico ou assimétrico. No caso do problema simétrico, a distância entre as cidades origem e destino é a mesma das cidades destino e origem. No caso do problema assimétrico, a distância da cidade A para a B é diferente da distância da cidade B para a A.



Exemplo de um 2-opt para o problema do caixeiro viajante.

Dada uma sequência de ligações entre as cidades — chamadas de **arestas** — que formam um passeio completo — quer dizer que é possível visitar todas as cidades que estão nessa sequência de arestas de modo que podemos partir de qualquer uma das cidades e retornar para ela —, o operador **2-opt** primeiro identifica as duas arestas (a, c) e (b, d) no passeio e as substitui por duas outras arestas (a, b) e (c, d), de modo que o resultado ainda seja um passeio completo com menor distância, ou seja, houve uma otimização da solução.

Uma busca local explora uma vizinhança de soluções do estado atual. Dessa forma, ele analisa quais são os estados que podem ser alcançados diretamente do estado atual. O método consiste em um processo iterativo que se move repetidamente do estado atual para um dos estados vizinhos para encontrar soluções melhores do que a melhor encontrada até agora. A estratégia mais simples é, a partir do estado atual, ir para o melhor vizinho. Essa solução, no entanto, não garante que a solução ótima local será encontrada. Existem estratégias bem mais sofisticadas para explorar a vizinhança e aumentar as chances de encontrar a solução ótima com um custo computacional reduzido.

Algoritmos genéticos

São uma técnica de busca aplicada a problemas de otimização baseados em alguns dos princípios de genética e seleção natural. Fazem parte de um ramo da computação conhecido como **computação evolutiva**. Semelhante ao que ocorre com os algoritmos de busca local, são aplicados para encontrar soluções ótimas, ou que satisfaçam determinadas condições que tornem as soluções aceitáveis. São utilizados com bastante frequência para resolver problemas de otimização e de aprendizado de máquina.

Atenção

Os algoritmos genéticos foram desenvolvidos, principalmente, por John Holland e David E. Goldberg (De Jong, 1975) e, desde então, foram aplicados em vários problemas de otimização com sucesso. Nesses algoritmos, em cada momento, temos um conjunto de possíveis soluções para o problema que está sendo resolvido. Essas soluções, então, são recombinadas e, além disso, passam por um processo chamado mutação (como na genética natural), produzindo novas soluções. E o processo é repetido nas demais iterações.

Os termos utilizados em algoritmos genéticos fazem referência ao processo evolutivo que ocorre na genética. Veja, a seguir, os principais termos dessa terminologia.

Cromossomos

É uma das soluções — não necessariamente ótima — para o problema em questão. Por exemplo, se as variáveis de decisão de um problema forem representadas por um vetor x de três posições, o cromossomo corresponde a uma das soluções que o vetor pode assumir, como (0,1,0) e (1,0,0).

Gene

É a posição de um elemento no cromossomo. Por exemplo, para o cromossomo (0,1,0), os genes correspondem às posições 1, 2 e 3.

Alelo

É o valor que um gene assume para um determinado cromossomo. Por exemplo, para o cromossomo (0,1,0), os alelos são os valores 0, 1 e 0.

Função de fitness

É uma função que recebe uma solução como entrada e produz a adequação da solução como saída. Em alguns casos, a função de fitness e a função objetivo podem ser as mesmas.

População

É um subconjunto de todas as soluções possíveis (codificadas) para um determinado problema.

Operadores genéticos

São funções que diversificam as soluções por meio dos seguintes processos:

- Busca na vizinhança de uma solução pela combinação de soluções — conhecido como cruzamento e normalmente referenciado pelo termo inglês *crossover*.
- Mutação que modifica uma solução usando algum recurso probabilístico.
- Seleção que escolhe a solução mais promissora para continuar a busca.

Veja, a seguir, a estrutura básica de um algoritmo genético.

Estrutura de um algoritmo genético.

O algoritmo precisa de uma solução inicial para o problema que será tratado por ele. Essa solução pode ser obtida por um método heurístico, seguindo algumas etapas. Conheça-as!

Solução trivial do problema

A primeira etapa consiste em explorar alguma característica específica do problema para obter uma solução trivial que satisfaça o conjunto de restrições. Essa é a forma mais simples de obter a solução.

Cálculo da função de fitness

O segundo passo é calcular a função de fitness para medir a importância dessa solução.

Cruzamento de soluções

Em seguida, o algoritmo aplicará o cruzamento entre as soluções. Isso significa que, se, em um dado momento, o algoritmo possui mais de uma solução, elas serão combinadas para gerar uma nova solução.

Mutação das soluções

Na etapa seguinte, algumas soluções passarão pelo processo de mutação.

Seleção das soluções

Depois, o algoritmo selecionará quais soluções continuarão a ser exploradas.

Verificação das condições de término

Por fim, o algoritmo verificará se as condições de término foram satisfeitas. Em caso positivo, ele vai terminar e apresentar a melhor solução. Em caso negativo, ele vai retornar para a etapa de cálculo da função de fitness e vai repetir a sequência de passos.

Como vimos na sequência de passos da estrutura dos algoritmos genéticos, eles realmente tentam imitar o processo de evolução genética. Além da natureza aleatória de exploração do processo de busca, eles apresentam vantagens sobre os algoritmos de busca local, pois também exploram informações históricas.

As principais vantagens e limitações dos algoritmos genéticos podem ser vistas a seguir Confira!

Vantagens

- Maior velocidade em comparação com os métodos tradicionais.
- Existência de recursos paralelos que podem ser explorados.
- Flexibilidade em termos de aplicações, podendo ser usados no contexto de problemas de otimização discreta e também para otimização contínua e funções multiobjetivos.
- Fornecimento de uma lista de “boas” soluções e não apenas de uma única solução.

Limitações

- Complexidade na implementação das etapas de cruzamento, mutação e seleção, afetando a convergência do algoritmo.
- Inexistência de garantias sobre a otimalidade da solução.

Neste vídeo, o especialista apresenta os conceitos de busca local e algoritmos genéticos, além de mostrar como identificar as técnicas que são mais apropriadas para cada situação. Assista para entender melhor!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Os algoritmos genéticos utilizam diversos conceitos relacionados ao processo de evolução biológica. A ideia é que, da mesma forma que acontece na natureza, o algoritmo possa evoluir até que obtenha a solução do problema para o qual ele é aplicado. Assim, selecione a opção correta a respeito das características dos algoritmos genéticos.

A

A seleção é a etapa responsável por combinar as soluções de modo a criar candidatos para a solução.

B

A seleção define a maneira como os indivíduos da população passarão por mutação de modo a aumentar as chances de obter melhores soluções.

C

O *crossover* considera quais são as mutações a que um indivíduo será submetido.

D

A mutação é responsável por selecionar os melhores indivíduos que devem continuar no processo de busca.

E

Depois de obter um conjunto de indivíduos, eles devem ser analisados pela função de fitness.

A alternativa E está correta.

A etapa de cruzamento combina as soluções; a de seleção escolhe a melhor solução para continuar o processo de busca; a de mutação faz modificações em uma solução na expectativa de potencializar o resultado da busca. A função de fitness é aplicada aos algoritmos genéticos para representar os principais requisitos da solução desejada de um problema, como qual é a solução mais barata e qual é a rota mais curta, por exemplo.

Questão 2

A inteligência artificial tem aplicações para diversas áreas. Essa diversidade de aplicações implica a necessidade de métodos mais eficientes, os quais se adaptem aos contextos dos problemas. Um desses métodos é o de busca local. Assim, selecione a opção que apresenta algumas das principais vantagens do método de busca local.

A

Utiliza menos memória que outras estratégias de busca, apesar de aumentar o tempo de processamento.

B

Possui um processamento mais eficiente que outras estratégias de busca, apesar de utilizar mais memória.

C

Quando combinado com outras estratégias de busca heurística, é eficiente no uso da memória, apesar de seu processamento ter um desempenho reduzido.

D

Utiliza menos memória que outras abordagens e é eficiente para buscar por uma solução em grandes espaços de estados.

E

É bastante eficiente em termos de processamento e sempre encontra a solução ótima global, desde que ela exista.

A alternativa D está correta.

Não é possível fazer afirmações sobre o tempo de processamento dos métodos de busca local, a não ser que seja um parâmetro da execução do algoritmo. Os métodos de busca local caracterizam-se pelo uso eficiente de memória, e quando é feita a combinação com outros métodos, a expectativa é que tenham um desempenho mais eficiente, apesar de não haver garantias. A estratégia de busca local faz uso eficiente da memória — geralmente, uma quantidade constante — e pode encontrar, de modo eficiente, soluções que satisfaçam determinados critérios de qualidade em espaços de estado grandes ou infinitos (contínuos).

Considerações finais

Você conheceu aspectos teóricos a respeito das técnicas de busca para a inteligência artificial. De fato, é necessário compreender muitos conceitos para estudar os algoritmos. Em contrapartida, os métodos incorporam muitos aspectos práticos do comportamento esperado de um agente inteligente. Assim, não é difícil fazer analogias para entender o comportamento dos algoritmos.

A diversidade de técnicas acontece pela adequação de cada uma delas a determinadas famílias de problemas. Entender essas estratégias e técnicas de busca nos ajuda a avançar nos estudos sobre inteligência artificial e perceber oportunidades de aplicações e de aperfeiçoamento de algoritmos.

Podcast

Ouça e entenda a importância dos conceitos e das aplicações de técnicas de busca para inteligência artificial.

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore+

Não pare por aqui! Assista ao filme que selecionamos para você:

O jogo da imitação, dirigido por Morten Tyldum, 2014. A obra trata da vida de Alan Turing na época da Segunda Guerra Mundial.

Referências

DE JONG, K. A. **An analysis of the behaviour of a class of genetic adaptive systems**. 1975. Tese (Doutorado em Computer and Communication Sciences) – University of Michigan, Ann Arbor, Michigan, Estados Unidos.

MCCARTHY, J. **What is artificial intelligence**. Stanford University, Computer Science Department, Technical report, 12 nov. 2007.

RUSSELL, S.; NORVIG, P. **Inteligência artificial**. 2. ed. Rio de Janeiro: Campos, 2004.