



INTELIGÊNCIA ARTIFICIAL APLICADA

AULA 3

Prof. Luciano Frontino de Medeiros



CONVERSA INICIAL

Nesta aula você estudará sobre os elementos relacionados à resolução de problemas de busca, à formulação de problemas e exemplos, bem como às estratégias de busca categorizadas em dois tipos: a busca sem informação e a busca com informação, finalizando com a escolha de funções heurísticas.

CONTEXTUALIZANDO

Resolver problemas é uma das atividades mais nobres desenvolvidas pelo intelecto humano (e mesmo outros seres vivos). Um dos tipos de agentes inteligentes mais utilizados se refere aos que estão relacionados à resolução de problemas. Encontramos tais problemas em diversas situações do dia a dia: programar uma linha de produção em uma fábrica, encontrar a rota mais curta entre um conjunto de cidades, programação de robôs que precisam operar buscando mínimo custo, etc. Várias são as situações nas quais o agente precisa aplicar a racionalidade para encontrar uma solução. Os problemas variam conforme o número de elementos envolvidos e as regras que estes propõem. Portanto, o conhecimento dos processos de resolução de buscas é um dos tópicos mais importantes, estudados desde o início da Inteligência Artificial, os quais serão abordados a partir de agora.

TEMA 1: RESOLUÇÃO DE PROBLEMAS POR BUSCA

Os agentes inteligentes mais simples, os reativos, conferem as suas ações ao mapeamento direto dos estados. Em ambientes nos quais tal mapeamento é grande demais, tanto em termos de memória quanto de tempo de processamento, os agentes não são eficientes. Nestes casos, os agentes que são baseados em objetivos saem-se melhor por considerarem os possíveis cenários e o quanto os resultados são bons ou não.

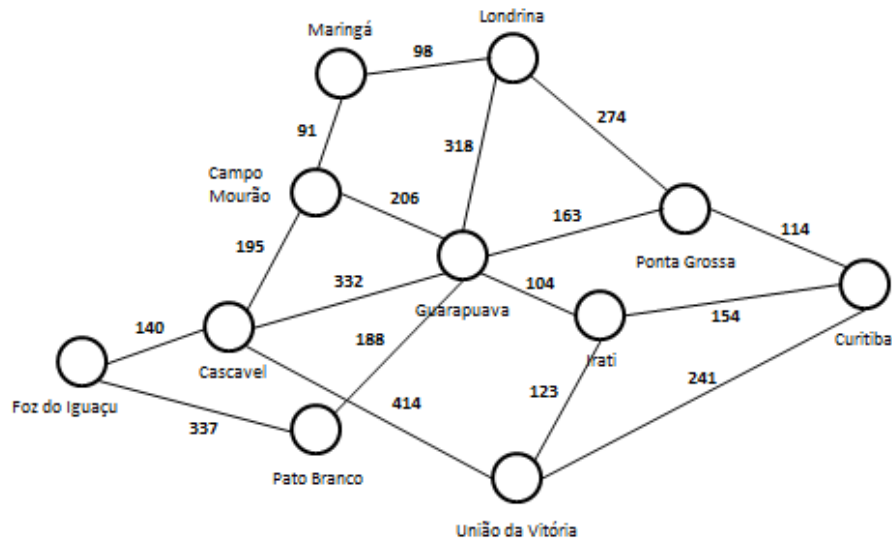


Os agentes de resolução de problemas tomam decisões sobre os próximos passos, encontrando as sequências de ações que resultam estados desejáveis (RUSSELL; NORVIG, 2004, p. 62). Por exemplo, um dos problemas presentes em várias situações é o de roteirização. Este problema está presente na área de logística e turismo, dentre outras. Quando se tem um conjunto de locais que precisam ser visitados, considerando a existência de rotas definidas para estes locais, um dos problemas típicos é como percorrer todos os locais considerando um custo mínimo ótimo. Este problema foi descrito inicialmente na IA como o problema do viajante.

Se considerarmos um mapa rodoviário, tal como o da Figura 1, se o objetivo é sair da cidade de Foz do Iguaçu em direção até a cidade de Curitiba, o agente deve escolher aquelas rotas ou cursos de ação que permitem chegar ao destino, e aqueles que não permitem chegar em um tempo aceitável podem ser rejeitados, simplificando assim a tarefa do agente. Dessa forma, a formulação de objetivos se baseia na situação atual do agente e na sua medida de desempenho, configurando o primeiro passo para a resolução do problema.

Aliado a este passo, a formulação de problemas é outro ponto essencial, pois é o processo no qual se decidem quais estados e ações deverão ser considerados na abordagem do problema, dado um determinado objetivo (RUSSELL; NORVIG, 2004, p. 62). Supondo que, no exemplo anterior, o agente esteja considerando as ações de dirigir de uma cidade até a próxima, os estados da rota adotada corresponderão a estados do problema.

Figura 1: Mapa rodoviário simplificado do Estado do Paraná



Supondo-se que o agente esteja em Cascavel, ele considera qual caminho pegar, podendo decidir ir por Campo Mourão, Guarapuava ou por União da Vitória. Entretanto, nenhuma destas cidades é o objetivo, que é o de chegar até Curitiba. Se o agente não tiver familiaridade com o trajeto, ele não saberá qual dos caminhos seguir. Em suma, o agente não poderá escolher um curso de ação porque não terá o conhecimento de qual estado resulta na execução de uma determinada ação. Caso ele não tenha nenhum conhecimento, se o agente não for programado para escolher aleatoriamente um caminho, ele ficará paralisado.

Se o agente possui um mapa, ele pode obter informações sobre os estados em que pode entrar e quais ações ele pode perfazer. O agente considera então as possíveis rotas a partir das cidades consideradas. Dessa forma, um agente que possua uma série de opções imediatas de valor desconhecido pode decidir o que fazer, avaliando primeiro as diferentes sequências de ações possíveis que irão levar a estados de valor conhecido, tendo condições de escolher a melhor sequência. O processo de procurar esta sequência é denominado de busca (RUSSEL; NORVIG, 2004, p. 62).

Um algoritmo de busca recebe na sua entrada um problema e apresenta na sua saída uma solução, descrita sob a forma de uma sequência de ações definidas. A partir da obtenção da solução, as ações recomendadas por ela são colocadas



em execução. O algoritmo para a resolução de problemas simples pode ser então descrito nos seguintes passos (RUSSELL; NORVIG, 2004, p. 63):

- formulação do objetivo e do problema;
- busca de uma sequência de ações que resolvem o problema;
- execução das ações uma por uma.

Quando a sequência se completa, o agente formula outro objetivo e então recomeça. É importante observar que na execução da sequência, ele não leva em conta as percepções, baseia-se na suposição de que a solução que encontrou na busca sempre irá funcionar.

Um problema pode ser definido de maneira formal em quatro componentes:

Um estado inicial: o estado no qual o agente começa. Por exemplo, se o agente do exemplo da viagem estiver em Foz do Iguaçu, o estado inicial pode ser descrito como $\text{Origem}(\text{Foz do Iguaçu})$.

Uma função sucessor: Dado um estado particular x , $\text{SUCESSOR}(x)$ retorna um conjunto de pares ordenados $\langle \text{ação}, \text{sucessor} \rangle$ em que cada ação é uma das ações válidas no estado x , e cada sucessor pode ser alcançado partindo-se de x . Como exemplo, no caso do estado inicial ser $\text{Origem}(\text{Foz do Iguaçu})$, a função sucessor para o problema retornaria

$\{ \langle \text{Destino}(\text{Cascavel}), \text{Origem}(\text{Foz do Iguaçu}) \rangle, \langle \text{Destino}(\text{Pato Branco}), \text{Origem}(\text{Foz do Iguaçu}) \rangle \}$

O estado inicial e a função sucessor definem o espaço de estados do problema, ou seja, o conjunto de todos os estados acessíveis, tendo como partida o estado inicial. O espaço de estados representa um grafo no qual os nós são os estados e os arcos entre os nós são as ações.

O teste de objetivo: determina se um estado é um estado objetivo. O objetivo do exemplo é o estado final $\text{Origem}(\text{Curitiba})$. Em alguns casos, um objetivo pode ser definido como uma propriedade abstrata e não por um estado ou



conjunto de estados específicos. Por exemplo, num jogo de xadrez, o objetivo é alcançar o xeque-mate.

A função de custo: também chamada de função de custo de caminho, que atribui um custo numérico a cada caminho. O agente irá escolher, portanto, uma função de custo que irá significar a sua própria medida de desempenho. No caso do agente do exemplo, a função de custo seria a distância em quilômetros (Figura 1). Há então um custo de passo para ir de um estado a outro.

Definidos os quatro elementos da formulação do problema, uma solução para um problema é um caminho que leva o agente desde o estado inicial até o estado objetivo. Considerando a qualidade da solução, a qual é medida pela função custo, uma solução ótima é aquela que apresenta o menor custo dentre todas as soluções possíveis (RUSSELL; NORVIG, 2004, p. 64).

Exemplos de Problemas

Para efeito de estudo, os problemas podem ser classificados em:

- Miniproblema: destina-se a ilustrar ou praticar métodos de resolução de problemas, podendo ter descrições concisas e exatas. Pode ser utilizado por diversos métodos para comparação de desempenho.
- Problemas do Mundo Real: são aqueles que não tendem a ter uma descrição concisa e exata, abstraída de situações da realidade. Em geral são problemas complexos, que podem ser divididos em problemas simples que possuem um método de solução conhecida. Esta estratégia é denominada de “dividir-para-conquistar”.

Miniproblemas

O primeiro miniproblema é o exemplo do robô aspirador, tratado no tópico sobre agentes inteligentes.



Tabela 1: Formulação do miniproblema do robô aspirador

Formulação	Descrição
Estados	O agente se posiciona em uma de 9 posições diferentes que podem conter sujeira ou não. A quantidade de espaços possíveis (contando que todos contenham sujeira) é de 9×2 estados mais 2 do estado inicial resultando em 20 estados.
Estado inicial	Robô na posição (2,2)
Função sucessor	Gera os estados válidos resultantes da tentativa de ação (Para-frente, Para-direita, Aspirar). Ver na Figura 3.
Teste de objetivo	Verificar se todos os quadrados estão limpos
Custo de caminho	Cada passo custa 1 unidade; o custo do caminho é o número de passos.



Esse miniproblema tem as posições discretas, sujeira discreta, limpeza confiável e não é desorganizado após o término da tarefa.

O *puzzle* (quebra-cabeças) de 8 peças consiste em um pequeno tabuleiro de tamanho 3×3 casas, com peças numeradas de 1 a 8 e um espaço vazio para permitir um deslocamento de uma peça. O objetivo do *puzzle* de 8 peças é atingir um determinado estado.

O *puzzle* de 8 peças pertence à família de quebra-cabeças deslizantes, utilizados com frequência em testes de algoritmos de IA. O quebra-cabeça de 8 peças possui 181440 estados acessíveis e pode ser resolvido com facilidade. O *puzzle* de 15 peças (um tabuleiro 4×4) possui cerca de 1,3 trilhão de estados. Já o *puzzle* de 24 peças (tabuleiro 5×5) possui aproximadamente 1025 estados, sendo ainda bastante difícil de resolver de forma ótima com computadores e algoritmos atuais (RUSSELL; NORVIG, 2004, p. 67).

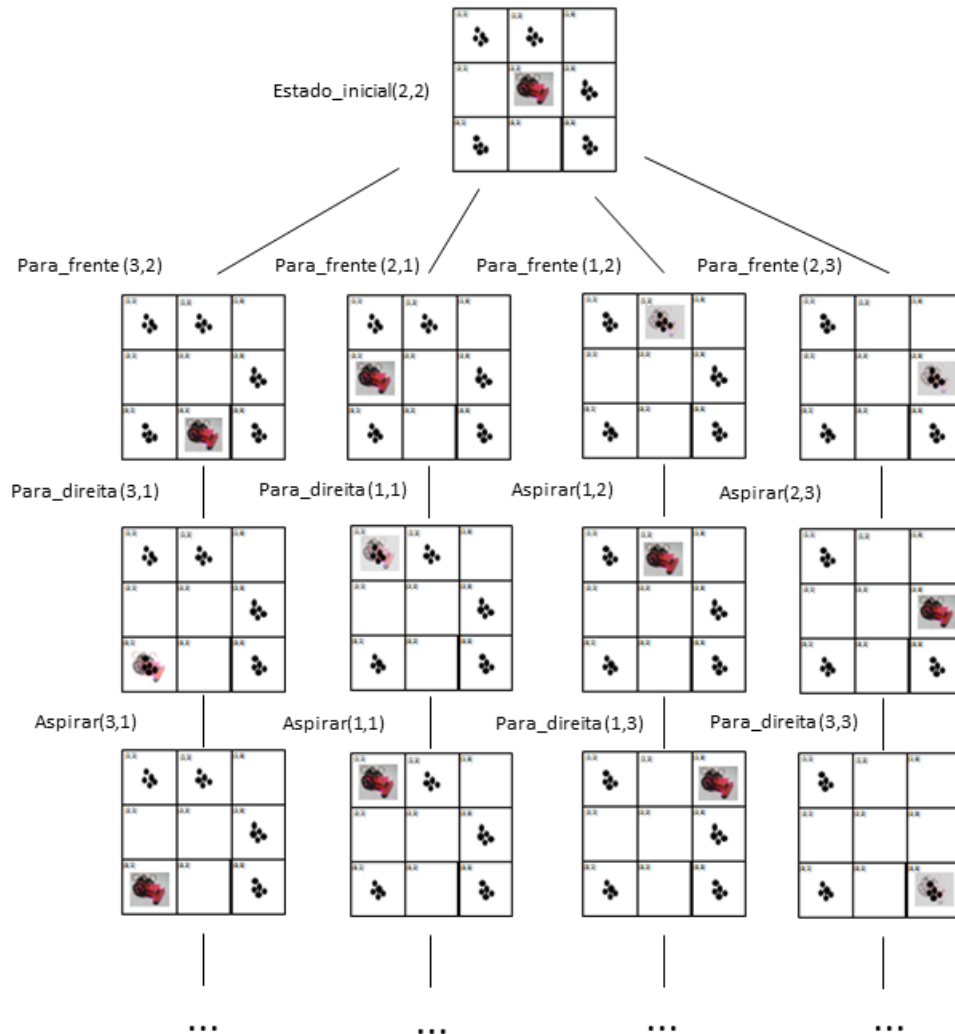


Tabela 2: Formulação do miniproblema do puzzle de 8 peças

Formulação	Descrição	Exemplo
Estados	Uma disposição específica das 8 peças sobre o tabuleiro 3x3.	
Estado inicial	Qualquer estado pode ser definido como o estado inicial (Ver Figura 4).	
Função sucessor	Gera os estados válidos para execução das quatro ações possíveis: Esquerda, Cima, Baixo, Direita.	
Teste de objetivo	Verifica-se que o estado alcançado é o mesmo que foi definido como objetivo.	
Custo de caminho	Cada passo custa 1 unidade; o custo do caminho é o número de passos.	

Descrevendo um problema do mundo real, o problema de roteirização ou roteamento baseia-se em uma série de pontos ou nós e ligações entre estes nós. Esta tarefa está presente em várias aplicações, tais como o roteamento de redes de computadores, planejamento de manufatura, operações militares, sistemas de planejamento de vôos e distribuição geográfica de produtos.

Figura 3: Expansão da árvore de estados do problema do robô aspirador



A formulação do roteamento pode ser aplicada posteriormente a qualquer caso real que atenda a estes pressupostos. Por exemplo, no caso de descobrir qual a melhor rota aérea entre duas cidades, os estados seriam as cidades pelas quais os vôos passam; o estado inicial seria a cidade de origem; a função sucessor retorna o próximo destino a partir de uma cidade sendo considerada; o teste de objetivo é saber se já se encontra na cidade destino; o custo do caminho pode ser o tempo de viagem, o valor monetário de cada trajeto entre cidades, dentre outros.

Figura 4: Expansão da árvore de estados do problema do puzzle de 8 peças

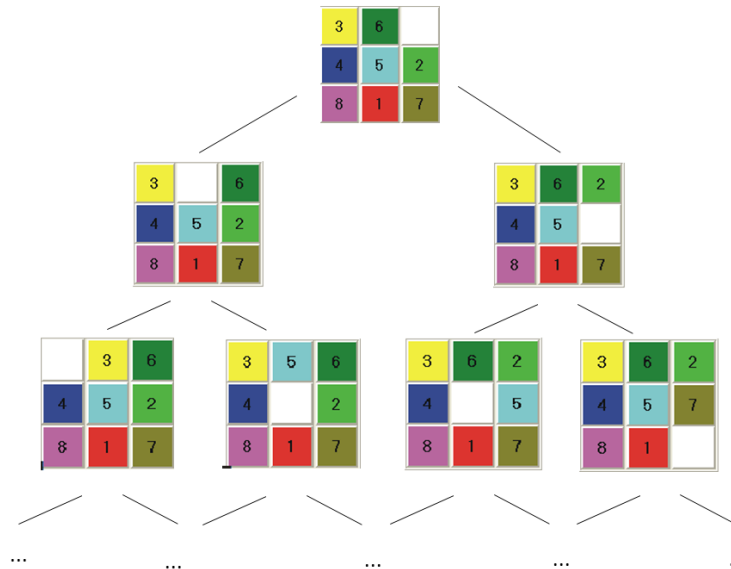


Tabela 4: Formulação do problema do roteamento

Problema de Roteamento	
Formulação	Descrição
Estados	Representação de uma posição ou nó
Estado inicial	Especificado de acordo com o problema
Função sucessor	Os nós ou posições que estão adjacentes
Teste de objetivo	Verificar se pela movimentação por meio dos nós, chega-se ao nó destino
Custo de caminho	Dado pelo somatório do custo de cada ligação entre os pares de nós

O problema do caixeiro viajante é uma extensão do problema de roteirização, em que todos os pontos de uma rede precisam ser visitados, em vez de apenas alguns. Este problema demonstra a explosão combinatória que pode acontecer,



mesmo o problema tendo poucas instâncias para resolução. A quantidade de possibilidades de rotas por todos os pontos é da ordem de $(n-1)!$ – fatorial de $n-1$. Para um conjunto de rotas com poucos pontos, o problema pode ser resolvido pela “força bruta” (calculando todas as rotas). Mas o problema cresce exponencialmente com a quantidade de pontos. Para o exemplo considerado do mapa do Paraná, no qual temos 11 cidades, a quantidade de rotas possíveis passando por todos os pontos é dada por aproximadamente $10!$ (fatorial de 10), quase 3,62 milhões de rotas. A tabela abaixo demonstra a complexidade deste problema, em que podemos identificar na segunda coluna o total de permutações possíveis de pontos a serem visitados e o tempo de execução, considerando que o cálculo de cada possibilidade leve 1 nanossegundo (1 bilionésimo de segundo). Pode-se identificar o salto que representa de 20 para 25 pontos em termos de alguns anos para milhões de anos de processamento!

Tabela 5: Demonstração da complexidade exponencial do problema do caixeiro viajante

Pontos ou nós	$(n-1)!$	Tempo
5	24	Insignificante
10	362880	Insignificante
15	87,17 bilhões	9 segundos
20	1,2.1017	3,9 anos
25	6,2.1023	19,6 milhões de anos

Outros problemas do mundo real podem ser enumerados, tais como o *layout* de circuitos integrados e processadores de computador, a navegação de robôs, sequência automática de montagem em fábricas robotizadas, projeto de proteínas e a pesquisa na internet (RUSSELL; NORVIG, 2004, p. 70).



Para saber mais...

O Problema do Caixeiro Viajante (chamado também de PCV) é um dos problemas mais difíceis de resolução. Assista ao vídeo produzido pela Sociedade Portuguesa de Matemática para entender um pouco mais a respeito deste problema.

Disponível em:

<https://www.youtube.com/watch?v=vKMyRj855A>.

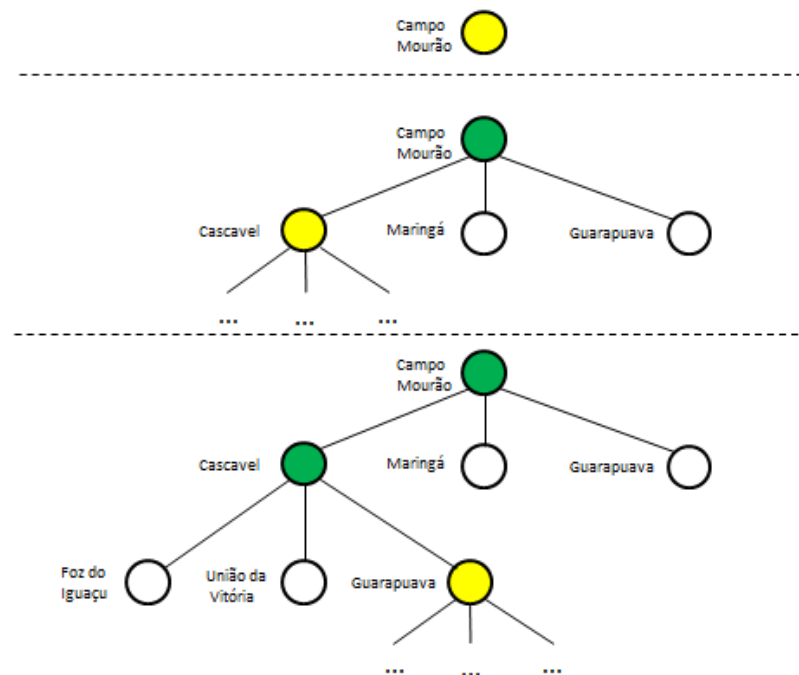


TEMA 2: BUSCA DE SOLUÇÕES

Para a resolução dos problemas expostos no tópico anterior, é necessário fazer uma busca no espaço de estados dos problemas. Nos problemas do robô aspirador e do *puzzle* de 8 peças descritos, não foi acidental a representação em forma de árvore nas figuras 3 e 4. Certas técnicas utilizam árvores de busca, que são geradas a partir do estado inicial e pela função sucessor, configurando assim o espaço de estados. Um nó de busca é a raiz da árvore, relativa ao estado inicial. Para a resolução, acontece então a expansão do estado atual nos estados possíveis a partir da função sucessor, que gera por sua vez um novo ramo ou conjunto de estados (Figura 5). No exemplo do mapa geográfico do Paraná, se estivermos no estado Origem(Campo Mourão), a expansão produziria um ramo com os nós Destino(Maringá), Destino(Guarapuava) e Destino(Cascavel).

Assim, o propósito da busca é fazer esta expansão de forma contínua, avaliando os nós gerados, escolhendo um especificamente e verificando se o nó é um estado objetivo ou não. Nesse processo, a escolha de qual estado expandir é determinada pela estratégia de busca (RUSSEL; NORVIG, 2004, p. 70).

Figura 5: Exemplo de expansão de uma árvore de busca para o problema da rota entre cidades



É importante notar a diferença entre espaço de estados e a árvore de busca. Os estados se referem aos nós possíveis do problema, mas pode existir árvores de busca com número infinito de nós.

Medidas de desempenho

Um algoritmo de resolução de problemas resulta uma solução ou uma falha. As falhas podem acontecer devido ao algoritmo ficar paralisado em um *loop* infinito e, assim, nunca retornar uma saída. A avaliação de desempenho de um algoritmo pode ser feita a partir de quatro aspectos (RUSSELL; NORVIG, 2004, p.73):

Completeza: o algoritmo de busca oferece a garantia de encontrar uma solução quanto ela existir?

Otimização: A estratégia encontra a solução ótima?

Complexidade de tempo: Quanto tempo é dispendido para encontrar uma solução?



Complexidade de espaço: Quanto de memória é necessária para a execução da busca?

TEMA 3: ESTRATÉGIAS DE BUSCA

As estratégias de busca dividem-se em duas classes: estratégias sem informação ou estratégias cegas e estratégias com informação ou busca heurística. As estratégias se diferenciam conforme a ordem de expansão dos nós da árvore de busca. As estratégias sem informação dividem-se em:

- Busca em extensão ou amplitude;
- Busca de custo uniforme;
- Busca em profundidade;
- Busca em profundidade limitada;
- Busca de aprofundamento iterativo;
- Busca bidirecional;
- Busca em Extensão ou Amplitude.

A busca em extensão ou amplitude é uma estratégia simples na qual o nó raiz é expandido inicialmente, depois os sucessores do nó raiz, depois os sucessores dos sucessores do nó raiz e assim por diante (RUSSEL; NORVIG, 2004, p. 74).

A busca em extensão é completa: se o nó objetivo estiver em uma profundidade finita d , a busca em extensão encontrará após a expansão de todos os nós mais rasos, considerando que o fator de ramificação b seja finito. O nó objetivo mais raso não significa que seja o nó ótimo. A busca em extensão será ótima se o custo do caminho for uma função não decrescente da profundidade do nó, como por exemplo, se o custo for o mesmo para todas as ações.

Se a árvore gera b nós no primeiro nível, o segundo nível subsequente gera também b nós, totalizando b^2 nós na árvore; no terceiro nível totalizará b^3 nós e assim sucessivamente. Se a solução estiver no nível d , o algoritmo se expandiria até o nível $d+1$, gerando então $(b^{d+1}-b)$ nós. A soma total dos nós será, então:



$$N = 1 + b + b^2 + b^3 + b^4 + \dots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

Utilizamos a notação O, à direita da equação, para indicar a complexidade do algoritmo de busca em extensão, onde indicamos o expoente mais significativo. Com um fator da ramificação b elevada ao expoente d, a complexidade se caracteriza como exponencial. Para poucos nós, o problema da busca em extensão é fácil de resolver. Mas se adicionarmos mais nós, a complexidade de tempo e memória aumenta significativamente.

Tabela 6: Análise da complexidade de tempo e espaço da busca em extensão ou amplitude

Profundidade (d)	Quantidade de nós	Tempo	Memória
2	1101	0,11 segundos	1 megabyte
4	111.101	11 segundos	106 megabytes
6	~10 ⁷	18,5 segundos	10 gigabytes
8	~10 ⁹	30,9 horas	1 terabyte
10	~10 ¹¹	128,6 dias	101 terabytes
12	~10 ¹³	35,2 anos	10 petabytes
14	~10 ¹⁵	3523 anos	1 exabyte
20	~10 ²¹	3,5 bilhões de anos	1 yottabyte

A tabela 6 mostra a análise da complexidade de tempo e memória, considerando um fator de ramificação de 10 nós para vários valores de profundidade d. Para o cálculo do tempo é considerado que 10000 nós podem ser gerados por segundo, e para o cálculo de memória cada nó exigindo 1000 bytes de armazenamento.

Ao lidar com um problema de profundidade 12, ainda levariam 35 anos para que a busca em extensão possa encontrar a solução. De forma geral, os problemas



de busca com complexidade exponencial não podem ser resolvidos por métodos sem informação, para qualquer instância, a não ser os problemas com instâncias menores.

Busca de Custo Uniforme

A busca de custo uniforme difere da busca em extensão por considerar a expansão do nó com o custo mais baixo. Este tipo de busca não se importa com o número de passos em um caminho, mas apenas com o custo total. Pode-se garantir a completeza desde que o custo em cada passo seja maior ou igual a um valor constante positivo pequeno denominado aqui de ϵ . Esta condição garante o caráter ótimo, significando que o custo é crescente à medida que se percorre o caminho. Se denominarmos C^* o custo da solução ótima, a complexidade do pior caso do algoritmo será de $O(b \lceil C^* / \epsilon \rceil)$. Quando todos os custos de passos forem iguais, o algoritmo volta à complexidade $O(bd+1)$.

Busca em Profundidade

A estratégia de busca em profundidade expande sempre o nó mais profundo na borda atual da árvore de busca. Na Figura 8 podemos ver como este algoritmo se desenvolve. A busca vai prosseguindo até o nível mais profundo da árvore, onde não existem mais nós sucessores. Após visitar os nós mais profundos, a busca retorna ao nível mais raso, que ainda possui nós não explorados (RUSSEL; NORVIG, 2004, p. 76).

Uma das vantagens da busca em profundidade é que ela possui requisitos de memória pequenos em relação às outras estratégias. Só é necessário armazenar um único caminho do nó raiz até o nó sendo visitado. À medida que os nós e seus descendentes são explorados, eles podem ser removidos da memória.

A desvantagem da estratégia de busca em profundidade é que ela pode descer um caminho muito longo (ou ainda infinito), podendo ficar paralisada, quando o nó objetivo poderia estar bem próximo do nó raiz da árvore de busca.

Busca em Profundidade Limitada

A abordagem da busca em profundidade limitada permite que atenuemos o problema das árvores muito grandes ou ilimitadas, delimitando a profundidade a



um valor máximo λ . Ou seja, os nós na profundidade λ são tratados como se não tivessem nós sucessores. Ele permite resolver o problema dos percursos infinitos, mas pode trazer um fator de incompleteza, ou seja, se $\lambda < d$, então o objetivo mais raso estaria além do limite imposto. A complexidade de tempo desta estratégia seria de $O(b\lambda)$.

Quando se tem algum conhecimento sobre o problema, pode-se impor um limite à estratégia de busca em profundidade. Citando um exemplo, o mapa geográfico do Paraná, apresentado na Figura 2, tem 11 cidades. Então poderíamos definir a profundidade 11 para a busca em profundidade de um percurso específico. Entretanto, se analisarmos melhor o mapa, podemos ver que uma cidade qualquer pode ser alcançada em 5 passos, podendo-se definir então o limite para o problema com $\lambda=5$. Mas na maior parte dos problemas não possuímos conhecimento da profundidade das soluções antes de resolvermos.

Busca de aprofundamento iterativo em profundidade

Esta estratégia é uma variação da busca em profundidade, que procura aumentar gradualmente o limite de profundidade a partir do nó raiz, depois 1, depois 2, e assim sucessivamente, até que encontre um objetivo. Isso deverá ocorrer quando o limite da profundidade alcançar d , a profundidade do nó objetivo mais raso. A estratégia de aprofundamento iterativo combina os benefícios da busca em profundidade e da busca em extensão. Os requisitos de memória também são modestos, exigindo $O(bd)$. Assim como na busca em extensão ou amplitude, o algoritmo é completo quando o fator de ramificação for finito, e ótimo quando o custo do caminho é uma função não decrescente em relação à profundidade do nó (RUSSELL; NORVIG, 2004, p. 79). A complexidade de tempo da busca em aprofundamento iterativo é melhor que a da busca em extensão.

De forma geral, a busca por aprofundamento iterativo em profundidade é o método de busca sem informação preferido quando há um espaço de busca muito grande e a profundidade da solução dentro da árvore não é conhecida (RUSSEL; NORVIG, 2004, p. 80).



Busca bidirecional

A noção na estratégia de busca bidirecional é a execução de duas buscas que acontecem simultaneamente, uma de forma direta, partindo do estado inicial como temos visto até agora, e outra que parte do objetivo, acontecendo a parada quando as buscas se encontram em um estado intermediário. A motivação principal é que uma busca que tenha uma expansão $bd/2 + bd/2$ é bem menor do que bd . Por exemplo, se o fator de ramificação $b = 10$ e a profundidade $d = 6$, numa busca em extensão padrão, teríamos um total de $N = 11.111.101$ nós (mais de 11 milhões de nós), enquanto que na busca bidirecional (considerando agora a profundidade $d = 3$, duas vezes) teríamos $N = 22.202$ nós. A complexidade de tempo da busca bidirecional é também igual a $O(bd/2)$.

A tabela 7 resume a comparação entre as estratégias de busca de acordo com os quatro critérios de avaliação (completeza, otimização, complexidade de tempo e de espaço).

Tabela 7: Tabela comparativa entre as estratégias de busca sem informação. O cabeçalho da coluna tem a sigla com as primeiras letras do tipo de busca (b é o fator de ramificação, d é a profundidade, m a profundidade máxima da árvore de busca e λ é o limite de profundidade. As anotações que estão em sobrescrito: a completa se b é finito; b completa se o custo do passo é $\geq \epsilon$, com ϵ positivo; c ótima se os custos dos passos são todos idênticos; d se ambos os sentidos utilizam a busca em extensão).

Critério	BE	BCU	BP	BPL	BAIP	BB
Completa?	Sima	Sima,b	Não	Não	Sima	Sima,d
Tempo	$O(bd+1)$	$O(b \lceil C^* / \epsilon \rceil)$	$O(bm)$	$O(b\lambda)$	$O(bd)$	$O(bd/2)$
Espaço	$O(bd+1)$	$O(b \lceil C^* / \epsilon \rceil)$	$O(bm)$	$O(b\lambda)$	$O(bd)$	$O(bd/2)$
Ótima?	Simc	Sim	Não	Não	Simc	Simc,d

Fonte: adaptado de Russell e Norvig (2004, p. 81).



Buscas com Informação

As estratégias de busca sem informação tendem a ser ineficientes na maioria dos casos. Estratégias de busca que utilizam algum conhecimento específico do sistema pode ser uma melhor opção mais eficiente. As estratégias neste caso apresentam uma abordagem denominada de busca pela melhor escolha. Esta busca considera um algoritmo que é uma especialização do algoritmo geral da busca em árvore, em que um nó é selecionado para expansão com base em uma função de avaliação $f(n)$. Na verdade, não há de fato uma melhor escolha, mas a escolha que parece ser a melhor, conforme a função de avaliação.

Há uma outra família de algoritmos com funções de avaliação diferentes, na qual um componente fundamental desta avaliação é uma função heurística $h(n)$. Por meio das funções heurísticas podemos aplicar o conhecimento relativo ao problema no algoritmo de busca. Por exemplo, podemos fazer com que a função heurística seja representada pela distância que o nó expandido está do objetivo. Existem dois tipos de busca com informação: a busca gulosa e o algoritmo A^* (chamado de “A-estrela”).

A busca gulosa é uma estratégia que tenta expandir o nó mais próximo à meta, na suposição de que levará provavelmente a uma solução de forma rápida, avaliando apenas a função heurística: $f(n) = h(n)$. No caso do mapa rodoviário do Paraná apresentado anteriormente, podemos utilizar a heurística da distância em linha reta. Apesar de não ser a distância pela estrada em quilômetros, o uso da distância em linha reta pode ser uma heurística útil para resolver o problema. Entretanto, isso não garante que se encontre a melhor solução quando aplicando para todos os casos.

A busca A^* é a estratégia mais conhecida de busca pela melhor escolha. A avaliação da busca combina o custo para alcançar cada nó, dado por $g(n)$, e o custo para ir do nó em questão até o objetivo $h(n)$ na forma:

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



Assim, $f(n)$ representa o custo estimado da solução de custo mais baixo passando pelo nó n .

Tabela 8: Tabela comparativa entre as estratégias de busca gulosa e A^* . O cabeçalho da coluna tem a sigla com as primeiras letras do tipo de busca (b é o fator de ramificação, m a profundidade máxima da árvore de busca. As anotações que estão em sobrescrito: a pode ficar presa em loops; b desde que não exista uma quantidade infinita de nós.

Critério	Busca Gulosa	A^*
Completa?	Não ^a	Sim ^b
Tempo	$O(bm)$	$O(bm)$ no pior caso $O(\log h^*(n))$ se o espaço de busca é uma árvore com um objetivo apenas
Espaço	$O(bm)$	$O(bm)$, expande todos os nós
Ótima?	Não	Sim

Fonte: adaptado de Russell e Norvig (2004, p. 97-101).

TEMA 4: FUNÇÕES HEURÍSTICAS

A definição de uma função heurística $h(n)$ irá depender da natureza do problema de busca. Vimos que uma boa função heurística requer uma heurística admissível, cujo comportamento nunca superestime o custo para se alcançar o objetivo (RUSSELL; NORVIG, 2004, p. 97). Assim, a modelagem de uma função heurística requer uma dose de bom senso no estudo das características do problema.

A função $h(n)$ do problema do mapa rodoviário é utilizada em grande parte dos problemas que possuem características geográficas ou espaciais, ligada ao conceito de distância euclidiana. A distância euclidiana é calculada a partir das coordenadas cartesianas dos pontos relacionados aos nós do problema.

Considerando a definição de uma função heurística para o problema do *puzzle* de 8 peças, podemos ver pela Figura 6, a seguir, que, a partir de um estado qualquer, as peças devem ser movimentadas para se alcançar o objetivo. Podemos então listar mais duas funções possíveis:

Número de peças fora de posição: Na figura 6, todas as peças no estado (a) estão fora de posição, ou seja, a função heurística assume o valor $h(n) = 8$.

Distância de Manhattan: calcular a distância em quadras de cada peça até a sua posição no objetivo. Na figura 13, as peças estão fora da sua posição conforme o seguinte: $h(n) = 3+2+2+2+2+2+2+1 = 16$ (a peça “1” precisa de 3 movimentos até o objetivo, a peça “2” precisa de 2 movimentos até o objetivo, e assim sucessivamente).

Figura 6: Estados referentes ao problema do *puzzle* de 8 peças: (a) um estado qualquer; (b) objetivo



Russell e Norvig (2004, p. 106) compararam o custo da busca do algoritmo A^* com as duas funções heurísticas anteriores com a busca por aprofundamento iterativo (BAI), gerando 1.200 problemas. Com a profundidade da árvore $d = 12$, a BAI obteve um custo de 3,644 milhões de movimentos, enquanto que a busca A^* obteve o custo médio de 227 para a heurística das peças fora de posição, e o custo médio de 73 para a função heurística utilizando a distância Manhattan.



SÍNTESE

Neste capítulo foram estudados os agentes de resolução de problemas, que precisam tomar decisões sobre os próximos passos a serem dados na sequência de ações. Ao se fazer a formulação do objetivo, o próximo passo é a formulação do problema. Geralmente o processo de encontrar uma solução significa perfazer uma busca no espaço de soluções possíveis. Um problema pode ser definido a partir do seu estado inicial, de uma função sucessor, o teste de alcance do objetivo e a função custo de caminho. Uma solução leva o agente do estado inicial até o objetivo. Uma solução ótima apresenta o menor custo de caminho dentre todas as soluções possíveis. Foram estudados diversos miniproblemas, tais como o do robô aspirador, o quebra-cabeça de 8 peças e o problema de roteamento. Para se fazer a busca, certas técnicas usam as árvores de busca. É necessário adotar estratégias de busca que possam ser sem informação e com informação.

REFERÊNCIAS

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. Tradução da 2. ed. Rio de Janeiro: Campus, 2004.

BITTENCOURT, G. **Inteligência Artificial** – Ferramentas e Teorias. Florianópolis: Ed. da UFSC, 1998.