FIAP ON

INTELIGÊNCIA ARTIFICIAL E COMPUTACIONAL

RESOLUÇÃO DE PROBLEMAS POR MEIO DE BUSCAS

2

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

2 RESOLUÇÃO DE PROBLEMAS POR MEIO DE BUSCA

2.1 Problemas e Soluções

Há diversas maneiras de se resolver um problema, assim como há diversos tipos de problemas. Geralmente, os problemas podem ser solucionados com um conjunto de ações que levam a um objetivo (**solução**). Imagine um jogador de xadrez (**agente**), ele deve seguir um conjunto de ações que leve à vitória. A cada jogada, ele deve escolher uma peça, que tem uma quantidade finita de possíveis movimentos. A melhor escolha eleva as chances de o jogador ganhar a partida.

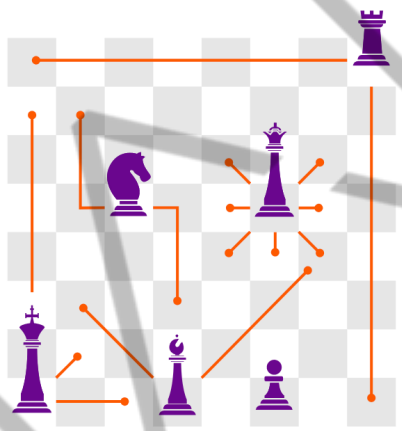

Figura 2.1. Possíveis movimentos das peças de um jogo de xadrez
Fonte: Google (2017)

Considere, como outro exemplo, um carro autônomo (**agente**) que deve se locomover até um destino final. Durante o percurso, seus sensores detectam uma série de objetos, como: ciclistas, pedestres, outros carros, postes, guias e a via. Baseado nesses reconhecimentos, o sistema autônomo (**inteligente**) deve escolher as melhores ações: como movimentar o volante, trocar de marcha, acelerar ou frear, dentre outras ações.

Quando as possibilidades (**ações**) para solução de um problema tornam-se quase inumeráveis, o espaço para se explorar uma solução deve ser considerado com técnicas que guiem a resolução do problema, levando em conta limitações de tempo, processamento e memória.

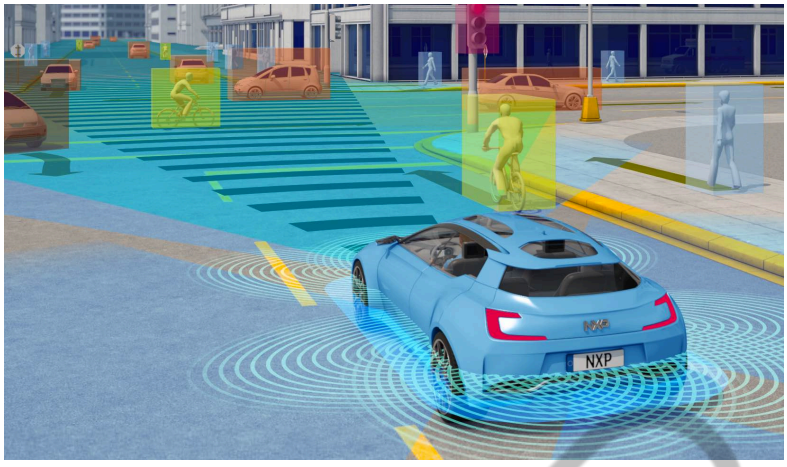

Figura 2.2. Ilustração do sensoriamento de um carro autônomo
Fonte: Google (2017)

Antes de explorar os métodos de busca de solução para um problema, é importante definir formalmente um problema.

2.1.1 Definição formal de um problema

Um problema pode ser definido formalmente com cinco componentes. Durante a definição, considere um agente que realiza entregas de produtos adquiridos pela internet partindo de um centro de distribuição (CD) até chegar ao local de entrega (LE), percorrendo o caminho (C) mais barato possível, de acordo com uma função de custo (FC).

Um problema é formalmente definido em cinco componentes (muito semelhante à definição de uma máquina de estados finitos, ou autômato):

1. Um **estado inicial** (s_0) a partir do qual o agente inicia. Tendo em vista o exemplo proposto, o agente inicia no CD, que é o s_0.
2. Uma descrição das **ações** (a_n) possíveis ao agente. Dado um estado s, AÇÕES(s) consiste em um conjunto de possíveis ações. Por exemplo, AÇÕES(s_0) consiste nas vias que partem do CD.
3. Um **modelo de transição** entre as ações, que define o próximo estado s_y, dada a escolha de uma ação, estando no estado s_x. É formalizado como RESULTADO(s_x, a) = s_y. Esses três componentes, estado inicial, ações e o modelo de transição compõem o **espaço de estados** do problema. O espaço de estados pode ser representado como um **grafo** dirigido, em que

os nós são os estados e as arestas são as ações. Nesse grafo, um **caminho** é uma sequência de estados conectados por ações (arestas).

4. Um teste de **objetivo**, que verifica, a cada estado, se o estado é o objetivo. Para alguns problemas, pode-se ter mais de um **estado objetivo**, que define a solução do problema. No exemplo dado, o estado objetivo é o local de entrega (LE) da mercadoria adquirida.

5. Uma **função de custo do caminho**, que atribui um valor numérico a cada caminho do grafo. Na função de custo do agente entregador a quilometragem, por simplicidade, pode ser considerada como função de custo do caminho, quanto menor o caminho, menor o custo.

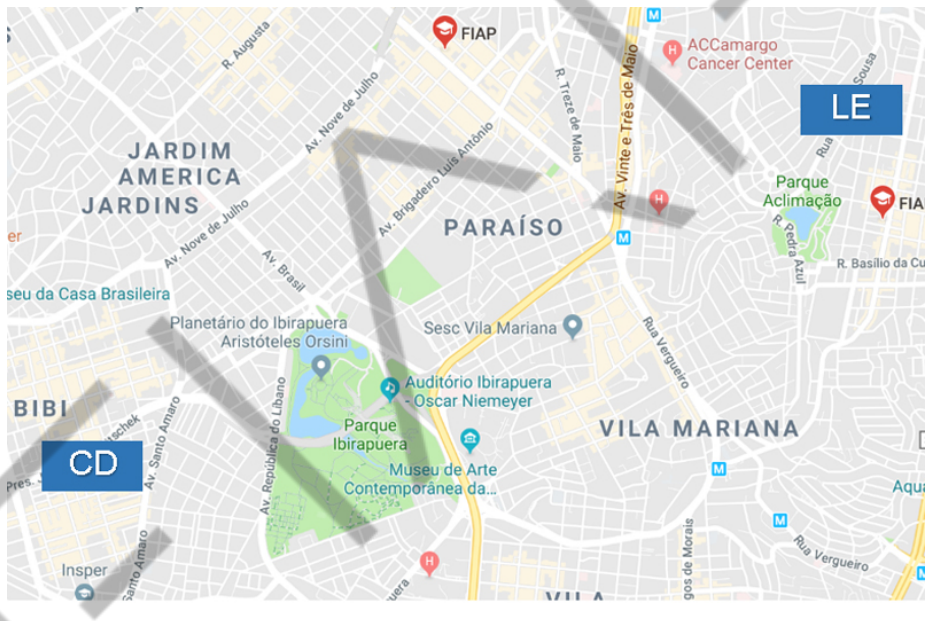

Figura 2.3. Mapa com localização do CD e LE para o problema de entrega de uma mercadoria
Fonte: Google (2017)

A figura ilustra o problema do agente que realiza entregas, partindo do CD, com objetivo de chegar ao LE com o menor custo do caminho. Como pode ser visto, há vários caminhos e, consequentemente, diversos custos. A busca pelo melhor caminho pode ser feita de diversas maneiras, como veremos mais à frente, neste capítulo.

A formalização realizada anteriormente é abstrata e possibilita a criação de um modelo simples para a resolução do problema. No mundo real, no entanto, há inúmeras variáveis a serem consideradas na modelagem do problema, aumentando as chances de sucesso do projeto.

No exemplo do agente entregador, além de considerar o caminho mais curto, a modelagem real deve levar em consideração outras variáveis, tais como:

- Tempo da viagem;
- Horário de trabalho do entregador, que deve se adequar à legislação trabalhista vigente;
- Velocidade das vias;
- Condições reais do trânsito nas vias;
- Número de cruzamentos;
- Dentre muitas outras variáveis.

Pode pensar em mais alguma variável para o problema do entregador? Lembre-se de que o veículo utilizado no transporte das mercadorias tem espaço limitado e, durante um caminho, o agente tem mais de uma entrega a realizar.

2.2 Métodos de busca por solução

Feita a formulação de um problema, deve-se trabalhar para encontrar a sua solução. De acordo com o modelo proposto, uma solução consiste em uma sequência de ações possíveis, partindo do estado inicial a um estado objetivo.

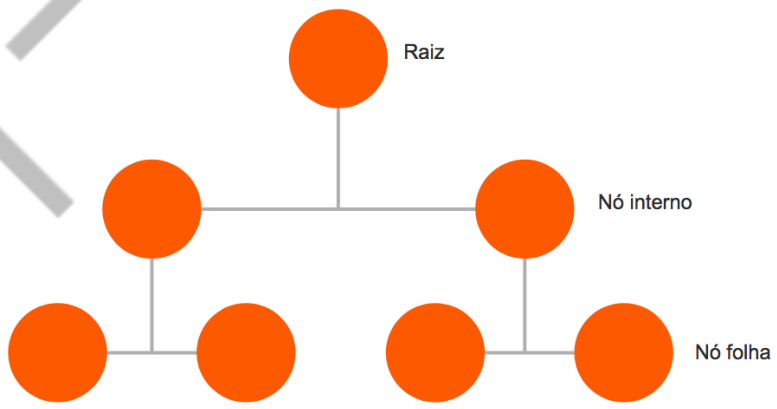

Figura 2.4. Exemplo de estrutura de dados em árvore
Fonte: Google (2017)

Cada possível solução de um problema forma o que chamamos de **árvore de busca**, originada do universo de possíveis soluções. Uma árvore consiste em uma

estrutura de dados não linear composta por nós (informações) organizadas hierarquicamente. Vale lembrar que, na computação, árvore é uma das principais estruturas de dados utilizadas e há diversos algoritmos eficientes de busca nessa estrutura.

Diversos algoritmos de ordenação, muito explorados em disciplinas introdutórias de Ciências da Computação, utilizam estruturas de dados de árvores. Por exemplo, um vetor de números inteiros pode ser armazenado em uma **árvore binária de busca**. Nessa árvore, os nós à esquerda do nó raiz contêm valores menores que o valor presente na raiz. Os nós à direita contêm valores maiores.

Essa restrição de ordenação é recursivamente aplicada a cada nó interno da árvore, até chegar a uma folha da estrutura. Isso garante uma complexidade computacional aceitável, mesmo quando a quantidade de números armazenados é muito grande. Se a árvore estiver balanceada, isto é, a quantidade de nós à direita e à esquerda da raiz é quase a mesma, a complexidade de busca é dada por $\log_2 n$, sendo n a quantidade de números armazenados. A figura exemplifica uma árvore binária de busca balanceada. Por exemplo, se armazenarmos 1 milhão de números, será necessário percorrer em torno de 20 nós da árvore para encontrar um número desejado ($\log_2 1000000 = 19.9315685693$).

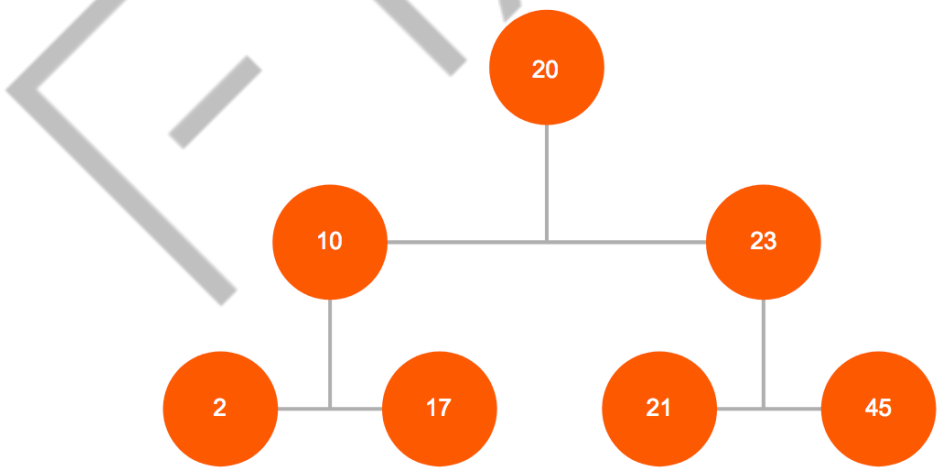

Figura 2.5. Árvore binária de busca balanceada
Fonte: Google (2017)

Dado um problema, basicamente pode-se optar por duas estratégias de busca: **busca sem informação** e **busca informada** com o uso de **heurísticas**. Nas próximas subseções serão apresentadas essas duas estratégias.

2.3 Busca sem informação

A busca sem informação, ou **busca cega**, é realizada sem informação adicional sobre os estados, além da definição do problema. A estratégia é gerar novos estados e verificar se um estado objetivo é alcançado ou não. Difere na ordem em que os estados são explorados: **busca em largura** ou **busca em profundidade**.

2.3.1 Busca em largura

A busca em largura (BFS – *Breadth-First Search*) é uma estratégia em que o nó raiz é verificado antes dos nós filhos do nó raiz. Essa estratégia é utilizada recursivamente para os nós internos da árvore (subárvores). A figura ilustra a busca em largura. Primeiro o nó A é visitado e seus filhos (B e C) são marcados para visita. Posteriormente, o nó B é visitado e seus filhos marcados para visita. A seguir, o nó C, filho de A é visitado e seus filhos marcados para visita. Esse procedimento é continuado até que todos os nós sejam visitados.

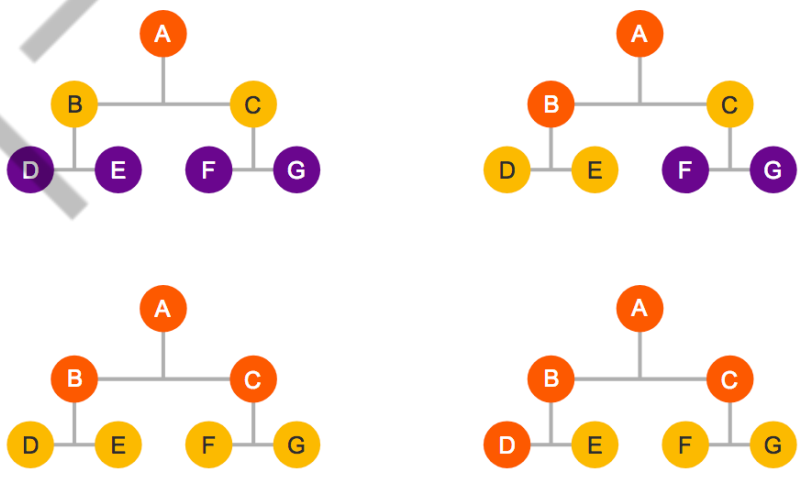

Figura 2.6. Ilustração de uma busca em largura
Fonte: Google (2017)

No exemplo, A é o nó raiz da árvore e seus filhos formam subárvores: B é raiz da subárvore e tem filhos D e E. Já C é raiz da subárvore formada por F e G. Esse é um exemplo de árvore binária pois cada nó tem dois filhos. No entanto, pode-se ter árvores n-ária, isto é, com n (fator de ramificação) filhos cada nó (n=2 para árvore binária).

2.3.2 Busca em profundidade

A figura ilustra a busca em profundidade (DFS – *Depth-First Search*). Veja a diferença em relação à busca em largura. Nesse procedimento, busca-se o nó mais profundo a partir da raiz. Dessa forma, visita-se o nó A, depois o B para se chegar ao D, que é um dos nós mais profundos da árvore. Retorna-se ao B para visitar o E, outro dos nós mais profundos. Esse procedimento continua até que toda a árvore seja coberta.

Nessa busca, toda uma subárvore à esquerda é visitada primeiro, para depois considerar a subárvore à direita. Caso a solução do problema fosse o nó C, ele só seria encontrado depois da visita aos nós A, B, D e E.

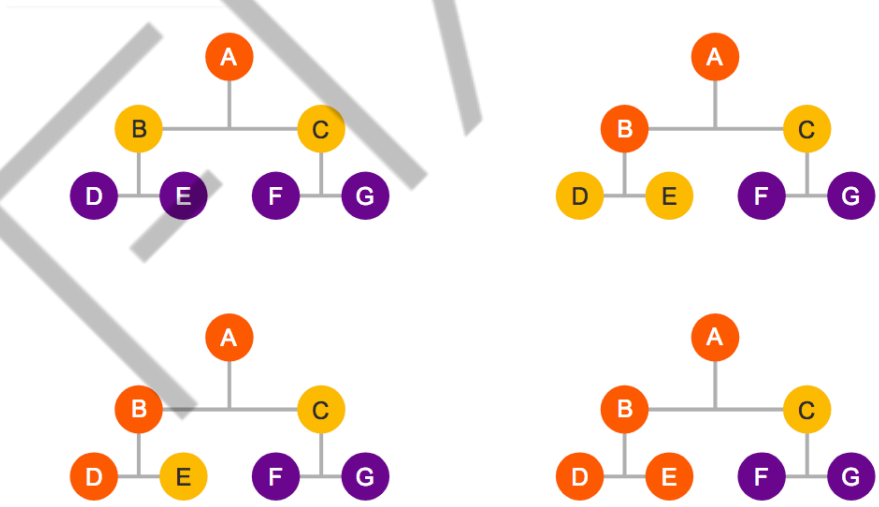

Figura 2.7. Ilustração de uma busca em profundidade
Fonte: Google (2017)

2.3.3 Comparação entre estratégias

Há diversas outras estratégias de busca não informadas, tais como:

- Busca com Custo Uniforme;
- Busca em Profundidade Limitada;
- Busca de Aprofundamento Iterativo e
- Busca Bidirecional.

Aqui vale citar quatro critérios utilizados para se escolher dentre as estratégias de busca:

1. **Completeza:** o algoritmo garante encontrar a melhor solução?
2. **Otimalidade:** encontra a melhor de todas as soluções possíveis ao problema?
3. **Complexidade de tempo:** quanto tempo para encontrar a solução.
4. **Complexidade de espaço:** quanto de memória é necessária para encontrar a solução.

Tendo esses quatro critérios em consideração, a tabela compara as estratégias de busca não informada. Na tabela, considere *b* como o fator de ramificação, *m* a profundidade máxima da árvore e *d* a profundidade da solução mais rasa (mais próxima da raiz). O(n) é a notação assintótica, que indica a complexidade de tempo ou espaço para valores muito grande de n.

Tabela 1. Comparação entre estratégias de busca.

Critério	Largura	Profundidade
1	Sim	Não
2	Sim	Não
3	O(b^d)	O(b^m)
4	O(b^d)	O(b^m)

Fonte: Google (2017)

Olhando para a tabela, parece que a busca em largura é a melhor opção, mas em termos de complexidade. A busca em largura pode levar muito tempo e ocupar muito espaço em memória para problemas cuja solução esteja a uma profundidade

alta (*d*). Já esse custo é assintoticamente o mesmo para a busca em profundidade, que depende de *m* (profundidade máxima da árvore).

2.4 Busca informada (heurística)

Já a busca informada é assim chamada pois se tem informação, dado um conjunto de possíveis estados, de qual é mais promissor, dado o estado atual. Esse conhecimento adicional é obtido por heurísticas, ou aproximações, utilizadas durante a busca.

Nas estratégias de busca informada, a cada passo utiliza-se uma **busca de melhor escolha**. A escolha de um nó a ser percorrido é feita com base na **função de avaliação** $f(n)$. Assim, dado um estado atual, a função é aplicada a cada possível expansão da solução com a finalidade de avaliar o custo. A expansão com menor custo é escolhida.

A diferença entre as estratégias de busca informada consiste na escolha da função $f(n)$. Como parte de $f(n)$, a maior parte das estratégias utiliza uma função heurística $h(n)$, que estima o caminho de menor custo do estado n ao estado objetivo.

2.4.1 Busca gulosa de melhor escolha

A busca gulosa expande a busca para o nó que seja o mais próximo do estado objetivo considerando apenas o caminho do estado atual ao próximo estado. Considerando o exemplo dado no início deste capítulo, do agente de entregas, se o agente está no CD, objetivando ir ao LE, ele pode utilizar como $h(n)$ a distância em linha reta para a próxima rua. A figura ilustra um possível cenário para o agente de entregas. Ir do CD ao A custa 1, do CD ao B custa 2 e do CD ao C custa 5.

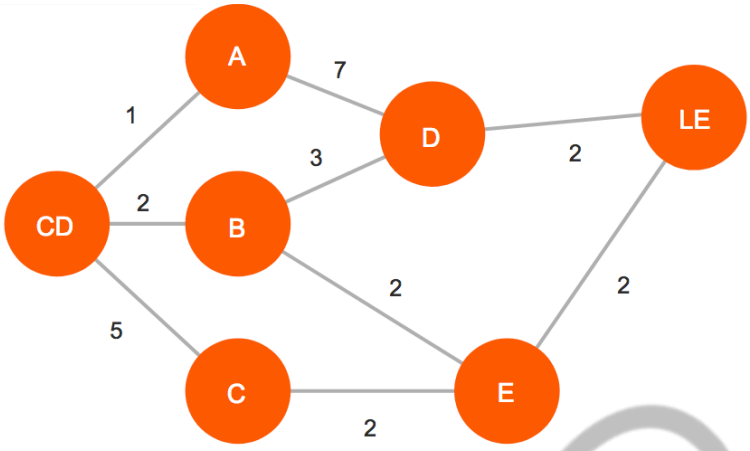

Figura 2.8. Possíveis caminhos entre o CD e o LE, para o problema do agente de entregas
Fonte: Google (2017)

A busca gulosa, estando no CD, considera o menor custo para a próxima expansão, isso levará ao nó A, que custa 1. A partir de A, a próxima expansão levará a D e, depois a LE. Esse caminho tem custo total de 1 + 7 + 2 = 10. Obviamente, esse não é o melhor caminho, que é CD – B – D – LE, com custo total de 2 + 3 + 2 = 7.

Nessa estratégia $f(n) = h(n)$, ou seja, a função de avaliação é igual à função heurística de escolher o menor custo a partir do estado atual.

2.4.2 Busca A* (minimizando custos)

A busca A* (“A estrela”) tem como função de avaliação $f(n) = g(n) + h(n)$, sendo $g(n)$ o custo para ir do nó inicial ao nó n e $h(n)$ o custo estimado para ir do nó n ao nó objetivo. Essa estratégia é uma combinação de aproximações heurísticas, como a busca em largura e o famoso algoritmo de *Dijkstra*. Ele é muito utilizado em aplicativos de rotas de deslocamento entre localidades e até na resolução de quebra-cabeças. Também é muito utilizado em jogos.

Considere:

Q = um conjunto de nós a serem percorridos;

P = conjunto de nós explorados;

S = o estado inicial da busca.

```
Inicialize Q com S
Repita
     Se Q está vazio
          Retorne "Falha"
     Obtenha nó n de Q, com melhor f(n)
     Se n é um estado objetivo
          Retorne "Solução(n)"
     Adicione n a P
     Para cada filho m de n, faça
        Adicione m a Q
```

Algoritmo 2.1. Algoritmo da busca A*
Fonte: Google (2017)

A complexidade espacial (de memória) dessa estratégia pode se tornar exponencial, pois todos os nós explorados devem ser mantidos em memória. Essa limitação do algoritmo pode ser reduzida com o uso do aprofundamento iterativo (IDA* - *Iterative Deepening A**).

Para finalizar a apresentação de diversas estratégias de busca, tanto cegas quanto informadas, que foram projetadas por cientistas da computação, temos a seguinte pergunta: seria possível um agente aprender para melhorar a busca? A resposta é sim, e baseia-se na ideia do **espaço de estados do nível meta**. Basicamente, a cada passo computacional da busca A*, armazenam-se erros cometidos para evitar que se cometa o mesmo erro futuramente, evitando gastos computacionais desnecessários, tornando a busca mais inteligente.

2.4.3 Funções heurísticas

Para finalizar este capítulo, apresenta-se, com o exemplo do quebra-cabeça de oito peças, a noção geral de heurística. Considere o problema em um tabuleiro de 3 por 3, com 8 peças, cujo estado objetivo é ter as peças arranjadas conforme ilustrado na figura.

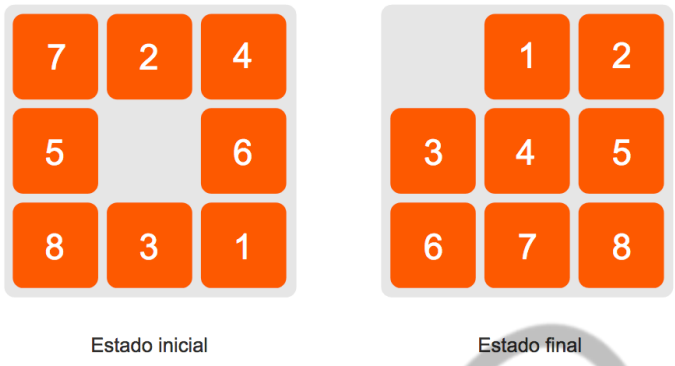

Figura 2.9. Exemplo do estado inicial e final de um quebra-cabeça de oito peças
Fonte: Google (2017)

O custo médio para solucionar o quebra-cabeça de oito peças é de 22 passos. O fator de ramificação é cerca de 3 (para uma peça que estiver no meio do tabuleiro são possíveis 4 movimentos; quando estiver em um dos cantos, 2; e quando estiver ao longo da borda, 3 movimentos). Se fosse feita uma busca exaustiva em uma árvore de profundidade 22, seriam necessários 31.381.059.609 estados!

Em vez de explorar todas as possíveis movimentações de peças, de maneira exaustiva, duas heurísticas ($h(n)$) poderiam ser utilizadas:

$h1$: número de peças fora do lugar. Assim, na figura, $h1 = 8$ para o estado inicial, pois todas as peças estão fora do lugar.

$h2$: soma das distâncias das peças de suas posições-objetivo. Dado que as peças não podem se movimentar na diagonal, utiliza-se a **distância de Manhattan**, ou **distância de quarteirão**, que é a soma das distâncias horizontais e verticais. Para o estado inicial, temos:

$$h2 = 3 + 1 + 2 + 2 + 2 + 3 + 2 = 18$$

Figura 2.10. Configuração do quebra-cabeça, em um estado n
Fonte: Google (2017)

Como exercício, quais seriam os valores para $h2$ e $h2$, dado o estado n, da figuraFigura 2.10?

REFERÊNCIAS

RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial.** Rio de Janeiro: Elsevier, 2013.

GLOSSÁRIO

Análise assintótica	A análise assintótica se preocupa em verificar como um algoritmo/solução se comporta quando os dados do problema são de grande magnitude.
Autômato	Modelo matemático de uma máquina de estados finitos. Esse modelo é definido como tendo diversos estados e possíveis transições entre esses estados. Serve para modelar uma máquina ou um computador simples.
Dijkstra	Cientista da computação holandês *Edsger Dijkstra* que solucionou (em 1959) o problema do caminho mais curto em um grafo dirigido.
Grafo	Modelo formado por vértices e arestas, que ligam os vértices. As arestas podem ter direção, indo de um vértice a outro, passando a chamar dígrafo (Grafo dirigido).