UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS

Trabalho Prático 1 - Busca

Alunos	Gabriel Ludwig Fonseca (202305184)
	Guilherme dos Santos (202402018)
Professor	Prof. Dr. Jomi Fred Hübner

Blumenau, 14 de Abril de 2024

Conteúdo

1 Introdução

A história da Inteligência Artificial (IA) é marcada por uma busca incansável pela criação de sistemas capazes de imitar e, em muitos casos, superar as capacidades humanas. Desde os primórdios da computação, os pesquisadores têm se dedicado ao desenvolvimento de algoritmos e técnicas que permitam às máquinas aprender, raciocinar e tomar decisões de forma autônoma. Esse avanço contínuo foi impulsionado pela crescente disponibilidade de dados e pela capacidade de processamento cada vez mais poderosa dos computadores modernos.

Nesse contexto, os problemas de busca desempenham um papel fundamental. A capacidade de encontrar soluções eficientes em espaços de busca complexos é essencial para uma ampla gama de aplicações em IA, desde a otimização de rotas em logística até a geração automática de planos de ação em sistemas autônomos. É nesse cenário que surgem os desafios como o "elevator logics", que exigem a aplicação de algoritmos de busca sofisticados para encontrar soluções ótimas em ambientes dinâmicos e complexos.

Diante da complexidade do problema dos elevadores, que envolve a coordenação de múltiplos agentes em um espaço de busca restrito, torna-se evidente a necessidade de desenvolver e aprimorar técnicas de busca que possam lidar eficientemente com essas condições desafiadoras. O objetivo deste trabalho é explorar e validar a eficácia de diferentes estratégias de busca na resolução do desafio dos elevadores, buscando não apenas encontrar soluções viáveis, mas também otimizadas, que minimizem o número de movimentos e maximizem a eficiência do sistema como um todo.

1.1 Descrição do Problema

O desafio conhecido como "Lógica dos Elevadores"é um exercício de raciocínio baseado em lógica, frequentemente utilizado como um jogo para testar habilidades mentais ou simplesmente para entretenimento. Pode ser acessado através do link da "Lógica dos Elevadores".

Figura 1: Elevators Logic Problem

O objetivo principal consiste em abrir os cinco elevadores, cada um contendo uma pessoa, e liberá-las. No entanto, os elevadores só irão se abrir se todos estiverem estacionados em algum andar entre o 21 e o 25, no nível 1 do jogo, ou entre o 21 e o 23, no nível 2. No nível 1, os elevadores começam em andares diferentes: 17, 26, 20, 19 e 31. Já no nível 2, suas posições iniciais são: 20, 20, 22, 24 e 21.

Além disso, há uma limitação de movimento para os elevadores: eles só podem subir até 8 andares de uma vez ou descer até 13 andares de uma vez. Essas ações podem ser repetidas quantas vezes necessário, contudo, somente dois elevadores podem executar a mesma ação simultaneamente.

O prédio possui 50 andares, numerados de 0 a 49. Portanto, se um elevador estiver em um andar superior ao 41, ele não poderá subir, pois isso o levaria ao andar 50, que não existe. Da mesma forma, se estiver no andar 12 ou abaixo, não poderá descer.

2 Algoritmos de busca

Após compreendermos a natureza do problema, o próximo passo é elucidar o processo de encontrar a solução ideal. Essa solução é representada pela trajetória, ou seja, a sequência de ações que conduz do estado inicial ao estado desejado. Os algoritmos de busca visam explorar diversas sequências de ações viáveis e retornar o caminho ou caminhos que levam ao estado final, podendo priorizar aqueles mais otimizados de acordo com critérios específicos.

É importante ressaltar que a eficiência de um algoritmo em uma determinada tarefa não garante sua adequação para alcançar a solução ótima em outro contexto. A complexidade variável entre diferentes algoritmos demanda uma análise minuciosa de seu desempenho no problema específico em questão.

2.1 Busca não informada

Os algoritmos de busca não informada não possuem capacidade de avaliar se um estado é superior a outro, ou se o trajeto que estão seguindo é o mais vantajoso. Sua única habilidade é distinguir se o estado atual corresponde ao estado final ou não. O que os diferencia é a sequência em que os nós serão expandidos.

2.1.1 Busca por largura

Na estratégia de busca em largura, após a expansão do nó raiz, todos os seus sucessores são explorados. Em seguida, os sucessores desses sucessores são considerados. O conceito-chave aqui é o uso de uma fronteira em forma de pilha, onde o primeiro elemento a entrar é o primeiro a sair. Uma analogia comum é a fila de um supermercado, onde as pessoas são atendidas na ordem em que chegaram.

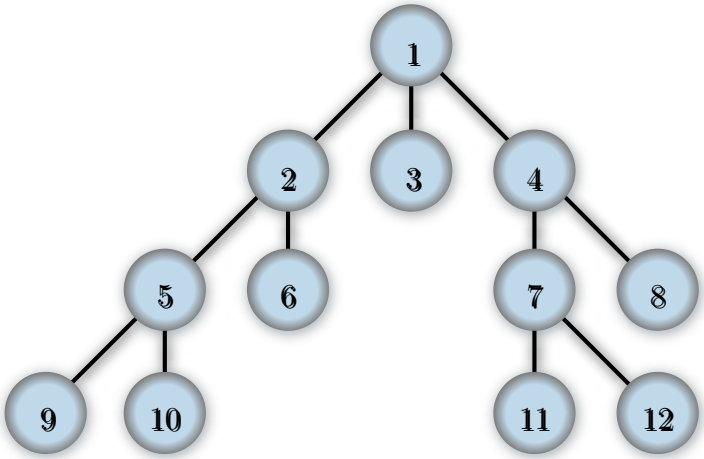

Figura 2: Busca em Largura. Fonte: Wikipedia

Essa estratégia é considerada completa, pois garante a exploração de todos os nós da árvore até encontrar a solução. Além disso, é ótima para o problema dos elevadores, pois encontra a solução mais próxima à raiz, ou seja, aquela que requer o menor número de ações. O tempo e o espaço necessários para encontrar a solução são proporcionais à ordem de grandeza do fator de ramificação b elevado à profundidade da solução d (o nível em que a solução está localizada), o que é representado por $O(b^d)$.

2.1.2 Buscar por profundidade

Na Busca por Profundidade, o algoritmo inicia pela raiz do grafo e explora o primeiro sucessor encontrado. Ele então avança para o primeiro sucessor desse sucessor, continuando assim até alcançar o ponto mais profundo, onde não existem mais sucessores. Para isso, utiliza-se uma estrutura de dados conhecida como pilha, na qual o último elemento inserido é o primeiro a ser retirado.

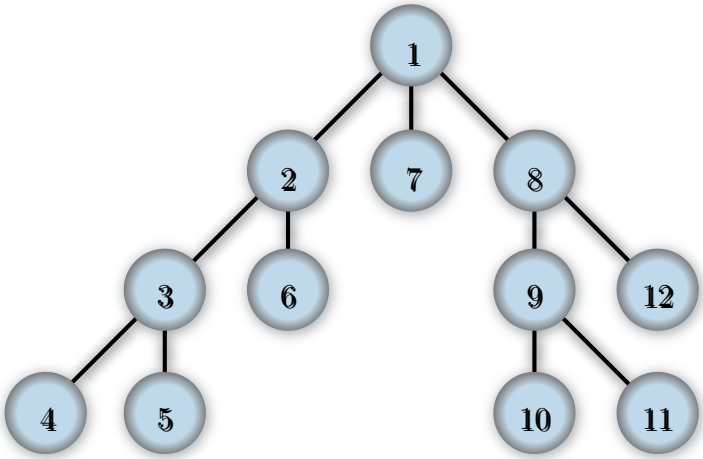

Figura 3: Busca em Profundidade. Fonte: Wikipedia

Essa abordagem de busca é considerada completa desde que não haja repetição de estados, ou seja, quando um estado já analisado não precisa ser revisado para evitar ciclos na busca. Para este trabalho, não ocorre repetição de estados. No entanto, essa técnica não garante a solução mais eficiente, pois pode encontrar caminhos que exigem um número maior de ações. O tempo necessário para encontrar a solução é aproximadamente $O(b^m)$, onde m representa a profundidade máxima de qualquer nó. Uma vantagem da busca por profundidade é sua eficiência em termos de memória: se um nó e todos os seus sucessores foram explorados sem sucesso, podem ser descartados da memória. Portanto, o uso de memória é proporcional a $O(b * m)$.

2.1.3 Busca em profundidade iterativo

Neste algoritmo, também é adotada uma abordagem com limite de profundidade. No entanto, a cada tentativa de resolver o problema sem sucesso, o algoritmo aumenta o limite de profundidade em uma unidade.

Essa estratégia é completa, pois o algoritmo continuará a aumentar o limite de profundidade até encontrar uma solução. Além disso, é ótima para o nosso caso, pois prioriza a busca pela solução mais próxima à raiz. O tempo necessário está relacionado à ordem de grandeza da última árvore testada, $O(b^d)$. A memória requerida está relacionada à ordem de grandeza O(b*d).

2.2 Busca informada

Na busca informada, o algoritmo tem a capacidade de avaliar se um estado é superior a outro utilizando uma função estimativa f(n). A ordem de expansão dos nós é determinada pelo nó que possui o menor valor para f(n) até o nó com o maior valor. A distinção entre os algoritmos de busca informada reside na maneira como a função f(n) é formulada.

2.2.1 Busca A*

Nesse algoritmo, o A* utiliza uma função de estimativa, que consiste na soma do custo do caminho percorrido até o momento com uma função heurística $f(n) = g(n) + h(n)$. Essa abordagem pode ser interpretada como o custo total da solução, priorizando a expansão do nó com o menor valor de $f(n)$ primeiro.

Para que o algoritmo A* seja ótimo, a heurística $h(n)$ deve ser admissível, o que significa que $h(n) \leq h^*(n)$, onde h^* representa o custo real do nó até o objetivo. O método A* é completo e ótimo se a heurística $h(n)$ for admissível. No pior dos casos, o tempo e o espaço necessários estão relacionados a $O(b^d)$, onde b é o fator de ramificação e d é a profundidade máxima da árvore de busca.

3 Estrutura do projeto

Os códigos podem ser acessíveis através do repositório no GitHub. O algoritmos escritos incluem busca em largura, busca de custo uniforme, busca em profundidade, busca em profundidade iterativa e A*. Foram utilizados diversos níveis de dificuldade que compõem o projeto, para ser mais evidente o tempo de execução dos algoritmos em determinados níveis.

3.1 Níveis de dificuldade

Considerando os diferentes níveis de dificuldade apresentados pelo problema, somos desafiados com duas variações a serem testadas. No nível 1, os elevadores devem iniciar nos andares 17, 26, 20, 19 e 31, com o objetivo do elevadores estarem entre os andares 21 e 25. No nível 2, os elevadores iniciarão nos andares 20, 20, 22, 24 e 21, tendo como objetivo meta somente os andares entre 21 e 23.

3.2 Heurísticas

Heurística em Inteligência Artificial (IA) refere-se a uma técnica ou abordagem que busca encontrar soluções para um problema de maneira eficiente, mesmo que não garanta a solução ótima. Em outras palavras, as heurísticas são regras práticas ou diretrizes que ajudam a orientar a busca por soluções em espaços de problemas complexos.

Ao contrário dos algoritmos exatos, que garantem a solução ótima, as heurísticas são mais rápidas e menos exigentes em termos de recursos computacionais. Elas são frequentemente utilizadas em situações em que encontrar a solução ideal é computacionalmente impraticável devido ao tamanho ou à complexidade do problema.

As heurísticas podem ser desenvolvidas com base na experiência humana, no conhecimento do domínio do problema ou em técnicas de aprendizado de máquina. Elas são amplamente aplicadas em áreas como otimização, planejamento, busca em espaços de estados, entre outras. Exemplos de heurísticas incluem o algoritmo A* para busca de caminhos em grafos, heurísticas de poda em algoritmos de busca em árvores de decisão, e heurísticas de seleção de variáveis em algoritmos de aprendizado de máquina.

A seguir são apresentadas as heurísticas implementadas no trabalho. Na primeira heurística utilizada no problema do trabalho a heurística é baseada no numero de ações que um elevador necessita para chegar na meta.

3.2.1 Heurística 1

A Heurística 1 é uma função utilizada em algoritmos de busca, como A*, para estimar o custo ou a distância entre um estado atual e o estado final desejado. Essa heurística é aplicada no contexto de um problema que envolve múltiplos elevadores e tem como objetivo minimizar a distância de cada elevador até o andar de destino mais próximo.

3.2.2 Heurística 2

A Heurística 2, denominada "Uso de Energia", é uma função utilizada em algoritmos de busca para estimar o custo energético associado à movimentação dos elevadores em um edifício. A heurística calcula o consumo de energia necessário para que cada elevador atinja seu andar de destino, considerando um custo de energia fixo por andar percorrido. Essa estimativa ajuda a orientar a busca em direção aos estados que requerem um menor consumo de energia, contribuindo para uma otimização do uso dos recursos energéticos.

3.2.3 Heurística 3

A Heurística 3, intitulada "Eficiência de Uso dos Elevadores", é uma função utilizada em algoritmos de busca para avaliar o quão eficientemente os elevadores estão operando em um determinado estado. Essa heurística calcula um escore de eficiência somando as distâncias de cada elevador ao andar de destino mais frequente. Quanto menor for o valor do escore de eficiência, mais próximo os elevadores estarão do andar de destino mais frequente, indicando um uso mais eficiente dos recursos disponíveis. Essa heurística é útil para direcionar a busca em direção aos estados que representam um uso mais eficiente dos elevadores, contribuindo para a otimização do sistema como um todo.

4 Complexidade dos Algoritmos de Busca

A tabela apresenta uma comparação entre diferentes estratégias de busca não informada e busca informada, destacando suas características principais em relação à complexidade no tempo e memória.

Estratégia de Busca	**Completa?**	**Ótima?**	**Complexidade de Tempo**	**Complexidade de Memória**
Busca em Largura	Sim	Sim	$O(b^d)$	$O(b^d)$
Busca em Profundidade	Não	Não	$O(b^m)$	$O(b \cdot m)$
Busca em Profundidade Iterativo	Sim	Não	$O(b^d)$	$O(b \cdot d)$
Busca A*	Sim	Sim	Varia de linear a exponencial	$O(b^d)$

Tabela 1: Comparação entre estratégias de busca não informada e busca informada

Essa tabela fornece uma visão geral das diferentes abordagens de busca, auxiliando na escolha da estratégia mais adequada para resolver problemas específicos em inteligência artificial e sistemas inteligentes.

5 Resultados

Os resultados obtidos para cada nível de busca implementada demonstram a importante escolha entre os diferentes tipos de algoritmos para resolução do problema em questão.

5.1 Desempenho

Você pode escolher uma estratégia de busca com base em quatro critérios:

1. Completeza: Um algoritmo de busca é considerado completo se, sempre que uma solução existir para o problema, ele conseguir encontrar essa solução.

2. Ótimo: Um algoritmo é considerado ótimo se consegue encontrar a melhor solução possível para o problema, dentre todas as soluções viáveis.

3. Tempo: Refere-se ao tempo necessário para que o algoritmo encontre a solução.

4. Espaço: Indica a quantidade de memória necessária para que o algoritmo encontre a solução.

Dentre essas estratégias vamos entender como nosso algoritmo desempenhou com as etapas de processamentos.

5.1.1 Nível 1

Para o nível 1, observou-se que todas as buscas foram capazes de encontrar uma solução viável, exceto a busca em profundidade que enfrentou problemas de memória. Notavelmente, a busca A* com a heurística adequada demonstrou um desempenho superior, explorando um número significativamente menor de nós em comparação com outras estratégias.

Estratégia de Busca	**Tempo aproximado**
Busca em Largura	6,1 segundos
Busca em Profundidade	*
Busca em Profundidade Iterativo	2,7 segundos
Busca A*	0,1 segundos

Tabela 2: Comparação entre os tempos de busca no nível 1

A otimização das heurísticas utilizadas permitiu uma melhoria adicional no desempenho da busca A*, destacando a importância de selecionar heurísticas adequadas para orientar a busca de forma eficiente. Esses resultados ressaltam a capacidade da busca A* em adaptar-se aos requisitos

específicos do problema, proporcionando soluções ótimas com eficiência computacional.

5.1.2 Nível 2

No nível 2, os padrões observados foram semelhantes aos do nível 1, com a busca A* continuando a se destacar em termos de eficácia e eficiência. A otimização das heurísticas, resultou em uma redução significativa no número de nós explorados, demonstrando mais uma vez a importância de ajustar as heurísticas para maximizar o desempenho da busca.

Estratégia de Busca	**Tempo aproximado**
Busca em Largura	22,9 segundos
Busca em Profundidade	*
Busca em Profundidade Iterativo	4,2 segundos
Busca A*	0,9 segundos

Tabela 3: Comparação entre os tempos de busca no nível 2

Em resumo, os resultados obtidos fornecem informações valiosas sobre a eficácia das diferentes estratégias de busca implementadas. Ao adaptar as heurísticas e selecionar cuidadosamente os algoritmos de busca, podemos resolver problemas complexos de forma eficiente e encontrar soluções ótimas em um espaço de busca limitado. Esses conhecimentos são essenciais para avançar na pesquisa em Inteligência Artificial e desenvolver sistemas inteligentes capazes de enfrentar desafios do mundo real com sucesso.

6 Conclusões

Com base nos resultados obtidos, podemos concluir que a escolha e otimização das estratégias de busca desempenham um papel crucial na resolução eficiente do problema dos elevadores. As diferentes abordagens de busca apresentaram resultados distintos em termos de tempo de execução e eficácia na busca da solução ótima.

No nível 1, observou-se que a busca em largura e a busca A* foram as estratégias mais eficientes, sendo capazes de encontrar uma solução viável em um tempo relativamente curto. A busca em largura explorou todos os nós da árvore de busca de forma sistemática, garantindo a completude da solução, enquanto a busca A* utilizou uma heurística adequada para direcionar a busca de forma mais eficiente, resultando em um tempo de execução ainda menor.

No nível 2, os resultados foram semelhantes, com a busca em largura e a busca A* continuando a se destacar em termos de eficácia e eficiência. A otimização das heurísticas permitiu uma redução significativa no tempo de execução da busca A*, demonstrando a importância de selecionar heurísticas adequadas para orientar a busca de forma eficiente.

Em resumo, os resultados obtidos destacam a importância de compreender as características e nuances de cada algoritmo de busca, bem como a influência das heurísticas na eficiência da busca. Ao escolher e adaptar as estratégias de busca de forma inteligente, podemos resolver problemas complexos de forma eficiente e encontrar soluções ótimas em um espaço de busca limitado.