Relatório

Inteligência Artificial

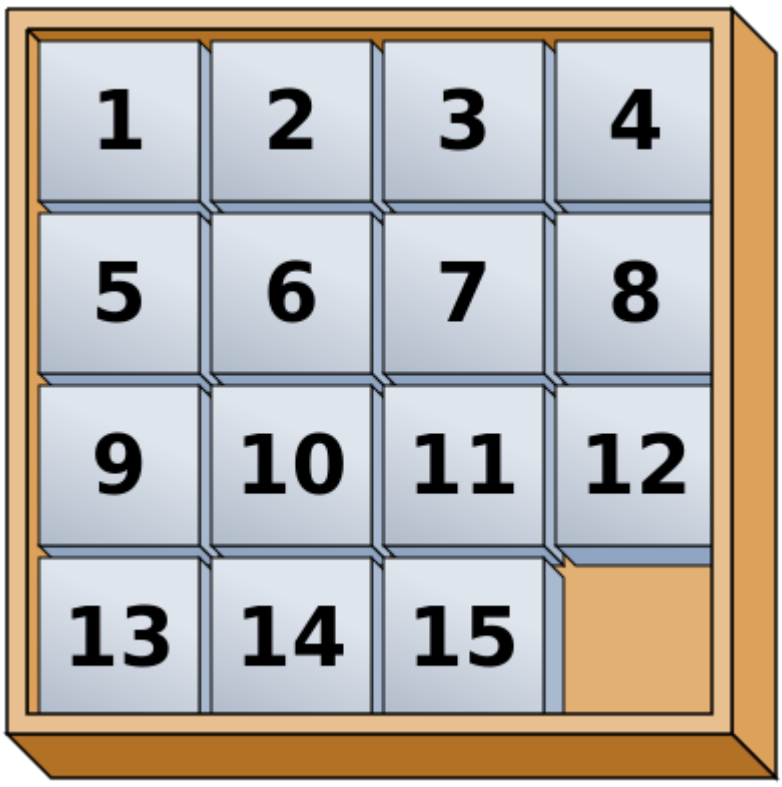

Illustration 1: Tabuleiro

Índice

1.1 Definição e descrição de um problema de busca

Em Inteligência Artificial o principal objetivo é possibilitar um sistema de resolver problemas de forma inteligente, isto é, tal como um humano poderia fazer. A habilidade para solucionar problemas, de facto, é frequentemente usada como modelo de inteligência para maquinas e homens.

Existem dois tipos de problemas. Um dos problemas pode ser resolvido por métodos determinísticos, ou seja, os problemas computáveis. No entanto, há poucos problemas do mundo real que possuem soluções computacionais. Então esses problemas são postos na segunda categoria, isto é, problemas não computáveis, e são resolvidos por meio de uma busca por uma solução.

Antes de elaborar um problema e utilizar os métodos de busca para resolver-lo é importante apresentar o objetivo a ser alcançado. Pois os objetivos ajudam a estruturar o programa através da restrição das finalidades que se pretende alcançar. Deste modo a primeira etapa na resolução de problemas é apresentar o objetivo.

1.2 Principais métodos utilizados para resolver os problemas de busca

Busca não informada:

Busca em largura (BL)

Busca de Custo Uniforme (BCU)

Busca de Profundidade (BP)

Busca Limitada em Profundidade (BLP)

Busca em Profundidade Iterativa (BPI)

Busca Bidirecional (BB)

Busca informada:

Busca Gulosa

Busca A*

2.1 Procura não guiada (blind/”cega”)

A busca cega não utiliza nenhuma informação sobre o domínio do problema para além das definições dadas. Apenas executam até encontrar uma solução ou mesmo uma falha. E o que faz é apenas distinguir o estado final dos outros estados.

Estas estratégias nem sempre funcionam da melhor maneira, podem falhar ao obter uma solução ou mesmo nem encontrar uma solução perfeita.

As principais estratégias de busca cega são : Busca em largura (BL), Busca de Custo, Uniforme (BCU), Busca de Profundidade (BP), Busca Limitada em Profundidade (BLP), Busca em, Profundidade Iterativa (BPI), Busca Bidirecional (BB).

2.1.1 Profundidade(DFS)

A Busca em profundidade (Deph-First Search), inicializa no nó raiz e pesquisa cada um dos seus ramos antes de voltar ao nó inicial, isto é , tende expandir sempre os nós de maior profundidade até conseguir o objetivo ou mesmo encontrar um nó sem filho. Dado que não encontra mais nós, a solução volta ao nó com maior profundidade ainda não visitado.

O algoritmo não é ótimo, já que dá privilégio aos nós mais profundos e poderá encontrar uma solução num nível mais profundo, quando a solução ótima se encontra num nó ainda não explorado numa profundidade menor.

O algoritmo de busca em profundidade não encontra necessariamente a solução mais próxima, mas pode ser mais eficiente se o problema possuir grande número de soluções ou se a maioria dos caminhos pode levar a uma solução.

Em relação à completude podemos dizer que não é completo, pois em alguns casos o algoritmo não nos consegue dar uma resposta em tempo útil, devido à árvore ter uma profundidade infinita, este método torna-se inútil uma vez que não encontra uma solução para o problema.

Complexidade Temporal: O(b^d)

Complexidade Espacial: O(b*m)

Onde b=fator de ramificação e m=profundidade máxima

2.1.2 Largura(BFS)

A Busca em Largura (Breadth-First Search), inicia-se no vértice raiz e explora todos os seus filhos, e para cada filho explora de novo os seus filhos,ou seja, explora todos os nós da mesma profundidade e só depois é que avança para a profundidade seguinte. E assim sucessivamente até chegar a uma solução pretendida.

Este algoritmo faz uma busca exaustiva, assegurando que todos os nós serão visitados, encontrando sempre uma resposta, isto é ser completo, e a solução é ótima pois dá a prioridade aos nós da mesma profundidade.

A utilização de memória nos estados de pesquisa em relação aos DFS os BFS gasta mais memória.

Complexidade Temporal: O(b^d)

Complexidade Espacial: O(b*m)

Onde b=fator de ramificação e m=profundidade máxima

2.1.3 Busca Iterativa Limitada em Profundidade

A Busca em Profundidade Iterativa (IDFS), tende combinar os benefícios das DFS e as BFS. Usa menos memória do que os BFS e consegue percorrer todo os espaços de busca, mas tem limite na profundidade máxima de caminho na árvore de busca.

O algoritmo corre várias instâncias de um DFS limitado em profundidade. Na primeira iteração a árvore gerada em profundidade limitada com limite igual a um, na segunda dois assim continuamente até ao limite máximo.

O método IDFS é ótimo mas não completa, pois depende do limite de profundidade.

Complexidade Temporal: O(b^L)

Complexidade Espacial: O(b*L)

Onde b=fator de ramificação e L=limite

2.2 Procura guiada

As procuras guiadas utilizam conhecimentos particulares ao problema a abordar com a intenção de avançar sempre um caminho que seja mais próximo da solução. Com este tipo de métodos de pesquisa podemos atingir um número menor de nós gerados e menos tempo para chegar a solução.

As principais estratégias de procura guiada são: Busca Gulosa, Busca A*.

2.2.1 O que é uma heurística ?

Quando fazemos um algoritmo, temos de ter cuidado com as propriedades na sua elaboração: ter um tempo de execução aceitável e ser ótima. E as heurísticas são criadas para encontrar soluções para o problema.

As heurísticas são procedimentos que ignoram informações que são do domínio do conhecimento especifico do problema para encontrar resultados.

Para muitos problemas, é possível estabelecer regras praticas para ajudar a reduzir a busca

Uma boa heurística para a estratégia de busca no estado de espaço de um problema é a estratégia conhecida como Subida da Encosta, também denominada como “subida da montanha” ou “hill climbing”

2.2.2 Busca Gulosa

O algoritmo de busca é um tipo de procura guiada que tem como objetivo minimizar o custo de uma solução através de uma heurística.

Este algoritmo é parecido com a Busca em profundidade : vai na mesma direção num caminho de árvore para procurar a solução mas pode mudar dependendo do custo dos nós ainda não visitados da árvore de pesquisa.

Como esta função não é monótona este algoritmo não é completa nem ótimo.

Complexidade Temporal: O(b^m)

Complexidade Espacial: O(b*m)

Onde b=fator de ramificação e m=profundidade máxima

2.2.3 Busca A*

A busca A*, é a combinação dos dois algoritmos: A busca gulosa e a busca de custo uniforme, algoritmo que encontra o caminho mais curto para um determinado nó.

Este algoritmo avalia o custo de nós através de:

- g(n) - custo real entre o estado inicial e o estado atual n,
- h(n) - custo estimado entre o estado atual n e o estado final., obtido através de uma heurística.

Então chegamos a equação f(n)=g(n)+h(n), onde f(n) significa custo estimado entre o nó *n* e o estado final.

Quanto menor for o valor de f(n), menor é o custo da solução. Por outras palavras, maior é a proximidade do estado final.

Esta busca é importante pois sempre que existe solução, ele alcança uma solução ótima.

Complexidade Temporal: O(b^m)

Complexidade Espacial: O(b*m)

Onde b=fator de ramificação e m=profundidade máxima

3. Descrição do problema de procura estudado e dos dois espaços de estado

O jogo do 15 é um quebra cabeças de quinze peças que contem números, figuras ou desenhos e um espaço vazio, representado por uma matriz 4*4.

Tem como objetivo movimentar a peça vazia de uma configuração inicial até chegar a uma configuração final, usando os movimentos (operadores): *cima, baixo, esquerda, direita.*

A partir de certas configurações iniciais com um objetivo pré-definido, não é possível chegar a uma solução. Isto deve-se ao facto de existir uma “sobreposição” de peças que geram paridade diferente em ambos os estados inicial e final de maneira que não seja possível, através de quaisquer movimentos, chegar ao estado desejado.

A fórmula utilizada para verificar a solvabilidade foi a disponibilizada nos slides (*Solvability of NxN-1 tile sliding problems*)[1]

4. Descrição da Implementação

O programa foi desenvolvido na linguagem C.

As estruturas utilizadas (*Queue, Stack e List*) foram disponibilizadas durante o estudo de Desenho e Análise de Algoritmos[2] para que não fosse consumido muito tempo na implementação de raiz e modificação/adaptação ao problema proposto.

Todos os algoritmos de pesquisa foram concebidos sobre estas estruturas.

O principal motivo pela escolha desta linguagem foi o facto de ser eficaz relativamente ao processamento e proximidade da gestão da memória.

Por outro lado, existe alguma repetição de código o que dificulta a leitura e compreensão e alguma dificuldade na depuração de erros que surgiram durante o desenvolvimento (alguns ainda por resolver).

5. Resultados

Estratégia	Tempo de exec.	Espaço	Solução?	Profundidade
DFS	300 seg. (limite)	75257 Nós ~6020560 Bytes	Não	-
BFS	11 seg.	13733 Nós ~1098640 Bytes	Sim	12
IDFS	2 seg.	1677061 Nós ~134164880 Bytes	Sim	12

	Estratégias

	A*			Greedy		
Heurísticas	**Tempo**	**Espaço**	**Depth**	**Tempo**	**Espaço**	**Depth**
Peças fora da pos.	0.001019 seg	280 Nós ~22400 Bytes	12	-	-	-
Manhattan Dist.	0.000413 seg.	59 Nós ~4720 Bytes	12	0.000152 seg	19 Nós ~1520 Bytes	12
Posições + Dist.	0.000181 seg.	25 Nós ~2000 Bytes	14	0.000261 seg	32 Nós ~2560 Bytes	16

6. Comentários finais e conclusões

Apesar dos testes de desempenho de cada estratégia terem sido executados num portátil antigo com hardware bastante limitado, foi possível comparar e distinguir as vantagens/desvantagens entre cada algoritmo de pesquisa.

Para este problema em específico foi utilizado apenas um *input* de profundidade/custo 12:

- Inicial: 13 11 15 4 8 9 1 5 12 14 0 2 7 10 3 6
- Final: 13 11 4 5 8 0 14 15 12 1 3 2 7 9 10 6

As combinações de estratégias e heurística com melhor resultado foram:

- A* com a soma das duas heurísticas (Nº de peças fora da posição e Manhattan Distance)
- Greedy com auxílio da Manhattan Distance

As pesquisas informadas demonstram ser mais vantajosas pelo facto de ajudarem na escolha correta do caminho durante a busca.

Nas pesquisas não-informadas, a ordem pela qual são gerados os nós sucessores através dos operadores podem influenciar o resultado final (e.g. DFS).

7. Referências Bibliográficas

https://rosettacode.org/wiki/15_Puzzle_Game

https://www.inf.pucrs.br/~pinho/LaproI/Vetores/Vetores.htm

https://pt.stackoverflow.com/questions/186611/d%C3%BAvida-sobre-valores-em-matriz

http://jeiks.net/wp-content/uploads/2013/05/Metodos_de_Busca.pdf

http://www2.ic.uff.br/~bianca/ia-20091/aulas/IA-Aula4.pdf

https://edisciplinas.usp.br/pluginfile.php/530273/mod_resource/content/2/Aula2-SolucaoPorBusca-2016-6pp.pdf

http://cee.uma.pt/edu/iia/acetatos/iia-Procura%20Informada.pdf

http://home.iscte-iul.pt/~luis/aulas/ia/Algoritmos%20de%20procura.pdf

http://www.rafaeldiasribeiro.com.br/downloads/IC1_6.pdf

[1] Solvability of NxN-1 tile sliding problems,
http://www.dcc.fc.up.pt/~ines/aulas/1819/IA/solvability.pdf

[2] DAA: Estruturas de Dados em C e Java,
http://www.dcc.fc.up.pt/~apt/aulas/DAA/1819/P/DAA1819_EstruturasDados.tgz

Ilustração 1, Tabuleiro do Puzzle do 15,
https://math.stackexchange.com/questions/635188/what-is-the-parity-of-permutation-in-the-15-puzzle