Inteligência Artificial

Tiago Moita e Milton Lopes

March 6, 2017

0.1 Introdução

0.1.1 O que é um problema de busca/procura?

busca/procura é um problema onde temos um conjunto de estados, dos quais um inicial,um final e o nosso objetivo será ir do estado inicial para o final através de uma função que vai mapear um estado num conjunto de novos estados.

O espaço dos estado pode ser representado por uma árvore de procura onde cada nó vai ser uma estrutura com pelo menos cinco componentes: estado,no pai,regra aplicada para gerar o no,profundidade do no(o caminho da raiz ate ao no),custo do caminho desde a raiz ate ao próprio no.

0.1.2 Quais são os métodos utilizados para resolver problemas de procura?

Métodos não informados de busca(busca cega):

1. Largura.
2. Custo uniforme.
3. Profundidade.
4. Limitada em profundidade.
5. Profundidade iterativa.
6. Bidirecional.

Métodos informados de busca:

1. Gulosa.
2. A*
3. Busca com memoria limitada.

0.2 Estratégias de Procura

0.2.1 Procura não guiada (blind - "cega")

Profundidade (DFS - Depth-First Search)

A busca em Profundidade expande todos os nos mais profundos da árvore primeiro, e vai perguntando a todos os nos em profundidade se são solução antes de passar para os nos gerados de um mesmo nível que o no a ser verificado.

Para problemas que têm muito soluções a busca em profundidade pode ser boa , pelo facto de ter boa chance de encontrar uma solução depois de explorar um espaço pequeno do espaço total de busca, mas tem o inconveniente de ser for usada para uma árvore com uma profundidade muito grande/infinita , pode demorar muito tempo "de um lado da árvore " enquanto que a solução pode estar "do outra lado da árvore".

A complexidade de espaço é somente os nós do caminho da solução + nos ainda não expandidos, e a complexidade temporal é $O(b^d)$.

onde b = fator de ramificação e m = profundidade máxima.

Largura (BFS - Breadth-First Search)

A busca em Largura expande todos os nos de um nível antes de expandir os nos do nível seguinte, e vai perguntando a todos os nos de um nível se são solução antes de passar para o nível seguinte.

Se o problema tiver solução a busca em largura garante que a encontra, e no caso de haver mais que uma solução ela garante que encontra a mais raza , ou seja a de nível mais baixo,ou seja a solução ótima.Utiliza se quando a profundidade da solução não é muito grande para uma dada instância pois o maior problema da busca em largura é quantidade de memoria utilizada, uma vez que é um método de busca não informado e vai gerar uma quantidade do nós de tamanho exponencial para este problema.

A complexidade de espaço e tempo são iguais $O(b^d)$

,pois todas as folhas da árvore tem que ser armazenadas ao mesmo tempo.

Busca Iterativa Limitada em Profundidade

A busca iterativa limitada em Profundidade funciona como se fosse uma junção entre a busca em profundidade e a busca em largura, ela vai gerando os nos filhos enquanto a profundidade de cada nó não for superior a profundidade corrente, e assim vai procurando profundidade a profundidade o nó objetivo, tem um particular muito bom que e o facto de ao não encontrar o no pretendido num nível x ,ela apaga todos nos gerados ate ao momento, claro que na iteração seguinte vai gera-los todos de novo , mas deste modo poupa muita memoria.

A estratégia é completa , mas ainda assim não e ótima pois ao iniciarmos a pesquisa damos de antemão uma profundidade máxima ,o que quer dizer que pode não encontrar a solução ótima se o nosso grafo tiver custos , caso não tenha ou seja "custo 1" para todos os caminhos , ai sim não haverá diferença pois a solução ótima será sempre a com menos profundidade.

A complexidade temporal será $O(b^l)$

onde l = limite de profundidade dada, e a complexidade espacial é O(b * l).

0.2.2 Procura guiada (que usa alguma heurística para orientar a procura)

As buscas informadas em geral utilizam conhecimento especifico do problema para encontrar a solução, e utiliza-se geralmente uma função de avaliação como por exemplo uma PriorityQueue em Java , para descrever a prioridade com que escolhemos os nos.

Gulosa

A busca gulosa utiliza uma função que determina o custo para atingir o estado final a partir de um determinado no (não custo exato, mas estimado e admissível), onde a esta função chamamos de heurística , e uma vez gerado os custos para chegar a solução final , podemos utilizar a dita fila de prioridade e escolher assim o nó mais próximo de objetivo, ou aquele que esta no caminho de menor custo.

A busca gulosa pode ser aplicada quando queremos chegar a uma solução ingenuamente , pois nem sempre será a ótima uma vez que podemos ter duas soluções distintas e se a nossa heurística não for a melhor podemos encontrar nós com a mesma heurística mas em níveis diferentes e a busca gulosa não faz a distinção dos níveis em que se encontram os nós, mas certamente gastará menos espaço e tempo que uma qualquer busca "cega".

A heurística escolhida para este problema "O Jogo dos 15" foi a Manhattan distance , pois achámos uma estratégia simples de ser implementada e bastante admissível ,uma vez que os nós neste jogo geram uma árvore com custos todos iguais de movimentação, e ao calcularmos a distancia segundo as movimentações admissíveis neste jogo , pareceu-me muito certo.

Tem complexidade temporal e espacial iguais $O(b^m)$.

A*

A busca A* é semelhante a busca gulosa, utiliza assim também uma função heurística para determinar o custo de determinado estado ao estado objetivo e também utiliza a profundidade de cada no , ou seja a busca A* vai metendo na nossa fila de prioridades os nós segundo o meu custo f = h(heurística) + g(distancia de cada no ate a raiz ou profundidade no dito problema "busca de custo uniforme") , e com isto consegue distinguir entre dois nos com a mesma heurística mas em níveis diferente , o que faz com que encontre certamente a solução ótima.

A estratégia A* é completa e ótima, com a restrição da nossa função heurística: nunca devemos super-estimar o custo real da melhor solução , deve ser "admissível".

A heurística escolhida para este problema "O Jogo dos 15" foi a Manhattan distance , pois achámos uma estratégia simples de ser implementada e bastante admissível, uma vez que os nos neste jogo geram uma árvore com custos todos iguais de movimentação, e ao calcularmos a distancia segundo as movimentações admissíveis neste jogo , pareceu-me muito certo.

Concluindo o A* pode ser utilizado quando queremos encontrar a solução ótima.

0.3 Descrição da implementação

Para este jogo escolhemos a linguagem java. Porque que escolhemos ? Certamente porque é a que nos sentíamos mais a vontade com, e também porque achamos que tinha vantagens por ter muitas funções já implementadas que nos iam dar jeito para o programa, e por ser uma linguagem multi plataforma , logo o que nós fizermos nas nossas maquinas certamente poderá rodar em qualquer outra maquina desde que a mesma tenha o JVM instalado.

Escolhemos as estruturas de dados baseando nos no problema que nos foi pedido, e tendo em conta sempre se eram eficiente para manipular os dados do problema, conseguimos correr todas as estratégias de busca com as ditas estruturas de dados , logo acho que foram nos suficientes e eficientes.

0.4 Resultados

Table 1: Configuração 1-Inicial

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

Table 2: Configuração 1-Final

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

Estratégia	Tempo(segundos)	Espaço	Encontrou a Solução?	Profundidade/Custo	Nos gerados
DFS			Não		
BFS	0.222s	228.269MB	Sim	12	37384
IDFS	0.367s	266.909MB	Sim	16	56164
Gulosa	0.004s	191.698MB	Sim	12	33
A*	0.004s	191.695MB	Sim	12	33

Table 3: Configuração 2-Inicial

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

Table 4: Configuração 2-Final

1	2	3	4
5	6	7	8
9	1	11	12
13	14	15	0

Table 5: Configuração 2

Estrategia	Tempo(segundos)	Espaco	Encontrou a Solucao?	Profundidade/Custo	Nos gerados
DFS			Nao		
BFS	0.008s	192.018MB	Sim	4	100
IDFS	0.007s	192.338MB	Sim	4	31
Gulosa	0.001s	192.018MB	Sim	4	12
A*	0.001s	191.018MB	Sim	4	12

Table 6: Configuração 3-Inicial

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

Table 7: Configuração 3-Final

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

Table 8: Configuração 3

Estratégia	Tempo(segundos)	Espaço	Encontrou a Solução?	Profundidade/Custo	Nos gerados
DFS			Não		
BFS	0.286s	258.070MB	Sim	13	72257
IDFS	0.633s	297.580MB	Sim	17	32505
Gulosa	0.004s	191.695MB	Sim	13	34
A*	0.007s	192.015MB	Sim	13	34

0.5 Conclusão

Em suma , podemos concluir que as estratégias de procura guiada são de longe melhores que as de procura não guiada uma vez que gastam menos memória e são mais rápidas a encontrar a solução. Por outro lado se a nossa prioridade for encontrar a solução ótima, podemos nos focar apenas na busca A* pois para um problema como este do jogo dos 15 onde o nosso espaço de busca é muito grande faz todo sentido preocuparmos-nos com a solução que gasta menos espaço e por sua vez demora menos tempo.

0.6 Referencias Bibliográfica

https://www.dcc.fc.up.pt/ ines/aulas/1617/IA/buscas$_n ao_i nformadas.pdf$
https://www.dcc.fc.up.pt/ ines/aulas/1617/IA/buscas$_i nformadas.pdf$