DCC028 - Inteligência Artificial

Trabalho Pratico 1: Busca em Mapas

Aluno: Romeu Junio Cunha de Oliveira **Matrícula:** 2012422971

Introdução

Neste trabalho prático foi implementado as versões de busca em grafos de quatro algoritmos conhecidos: *Interactive Deepening Seach, Unform Cost Seach*, *Best First Seach* e *A** utilizando como heurísticas as funções *Manhattan Metric* e *Octile Distance*.

O algoritmo *Interactive Deepening Search* realiza uma iteração sobre um outro algoritmo de pesquisa chamado *Depth-Limited Search*, que mistura os pontos fortes da busca em largura (*Breadth-First Search*) e da busca em profundidade (*Depth-Fist Search*). A cada rodada é incrementado a profundidade na qual o *DFS* irá realizar a busca. Os algoritmos *UCS*, *BFS* e *A** possuem a mesma implementação de busca em grafos. Neste modelo, é utilizado duas estruturas de dados: a lista **ABERTO(A)** e **FECHADO(F)**. Em *F* é armazenado os nós explorados (que já foram expandidos), e em *A* é armazenado os nós que foram expandidos, porém ainda não foram explorados. Ambos algoritmos expandem o nó de **menor custo**, é aí que eles diferem. Para o *UCS* o menor custo é calculado considerando o custo para chegar do nó início até ele. Para o *BFS* o menor custo é baseado apenas na função heurística, que no caso deste trabalho prático foi considerado apenas a heurística *Octile Distance*. O menor custo do algoritmo *A** é calculado considerando o custo para chegar ao nó do nó inicio até ele somado com a função heurística.

O algoritmo IDS é completo, porém só é ótimo quando o custo das arestas é constante. O algoritmo UCS é completo e ótimo (*Dijkstra*). O *BFS* não é ótimo pois é um algoritmo guloso, mas esta versão é completa por não entrar em loop. O *A** é sempre completo, e é ótimo dependendo se a heurística utilizada é consistente ou não.

Implementação

Python foi a linguagem utilizada para implementação dos algoritmos descritos (versão 3.6). Para execução do programa foram criados quatro arquivos *shell*: São eles: ***ids.sh, ucs.sh, bg.sh*** *e* ***aestrela.sh***. Ambos recebem como parâmetros o mapa para construção do grafo e as coordenadas x e y dos nós de início e fim. Para o algoritmo *A** um parâmetro adicional: (**1**: para heurística Manhattan e **2**: para a heurística Octile)

Exemplos de execução:

./ucs.sh map1.map 0 0 255 255

./aestrela.sh map2.map 125 125 50 50 1

Antes de começar a explicar o funcionamento do programa em si, algumas explicações sobre algumas escolhas precisam ser feitas. Primeiramente para a lista aberto decidi utilizar a estrutura ***heap queue*** (um heap com fila de priodidades), que na minha opinião é a estrutura perfeita para lidar com a lista *A.* Pois vemos que todos os algoritmos podem ser implementados como um único módulo de pesquisa (*Search.py*) e que diferem basicamente em como *A* é gerenciada, ou seja, sempre escolho e o expandir o nó de **menor custo.**

No caso do algoritmo *IDS* o **menor custo** não precisa ser considerado, dado que ele é uma combinação de busca em largura com busca em profundidade.

No *UCS* o **menor custo** para chegar até o nó n é somado o custo desde o nó de origem até o nó n.

No *BFS* o **menor custo** é dado apenas pela heurística *Manhattan Metric.*

No *A** o **menor custo** é dado pelo custo de chegar até o nó somado com a heurística (*Manhattan* ou *Octile*).

Então para obter este item de menor custo eu tenho uma complexidade O(1) e para inserir a lista *A* eu tenho a complexidade de ordenar um heap que no pior caso é O(log n).

Módulos

- Node: Classe de nó, que possui todas as informações necessárias pelo programa para reconstruir o caminho percorrido;
- Cost: Classe de custo, implementa a função de **menor custo;**
- BuildMap: Carrega o mapa contido no arquivo texto, e o armazena em uma *hash table* (em pyhon usado *dict*)
- Problem: Classe que descreve o problema, contendo mapa, estado inicial e estado objetivo;
- Queue: Implementa a lista *A;*
- BuildGraph: Valida o caminho no mapa, expandindo apenas os filhos válidos cumprindo as regras estipuladas, atribuindo o **menor custo** de acordo com o algoritmo selecionado;
- Tests: Execução de testes;
- Main: Classe principal que constrói o problema e executa o algoritmo selecionado.
- Search: Método de pesquisa genérico implementa a busca em grafos.

Implementação do algoritmo de pesquisa (Search)

Recebe como parâmetro o problema, ao executar o IDS *Search* recebe mais um parâmetro que é o limite. Inicializa as estruturas de dados: *F, A* com o estado inicial do problema e *path* (caminho percorrido). Verifica se o estado inicial é igual ao objetivo e retorna caso positivo. Verifica se o limite. A variável *cutoff_ocurred,* é indica para o algoritmo *IDS,* se chegou ao limite da profundidade da busca e não encontrou a solução, inicialmente inicializada como falso. Enquanto a lista *A,* não estiver vazia, selecionamos o item de menor custo. Verificamos se este item é solução retornando o resultado quando verdadeiro. Para o *IDS* quando o limite é igual a profundidade do nó, é atribuído o valor verdadeiro para a variável cutoff_ocurred. Caso não seja solução, expandimos o nó e o adicionamos na lista *F*, e para cada filho deste nó se ele não está na lista *A* ele é adicionado. Se o nó já está na lista *A* e estivermos executando o algoritmo IDS não é feito nada. Se o IDS não está sendo executado verificamos se o filho possui custo inferior ao que já está na lista *A* caso positivo ele é substituído pelo filho. Quando a lista *A* está vazia para *IDS* e a variável cutoff_ocurred possui o valor verdadeiro, significa a solução não foi encontrada na profundidade passada como parâmetro para o algoritmo. Se o valor é falso significa que não existe um caminho para a solução. Para os demais algoritmos a variável cutoff_ocurred sempre terá o valor falso retornando caminho inexistente para a solução.

Heurísticas

A heurística *Manhattan Distance não é consistente nem admissível.*

Por exemplo assumindo que existe o caminho válido pela diagonal, o custo para ir de (0,0) até (1,1) é 1.5. Esta heurística informa que o custo seria 2.

A heurística *Octile* é consistente e admissível

Considerando um mapa sem obstáculos, esta heurística realiza o menor caminho, pois ele caminha tudo que consegue pelas diagonais e o restante pelas laterais.

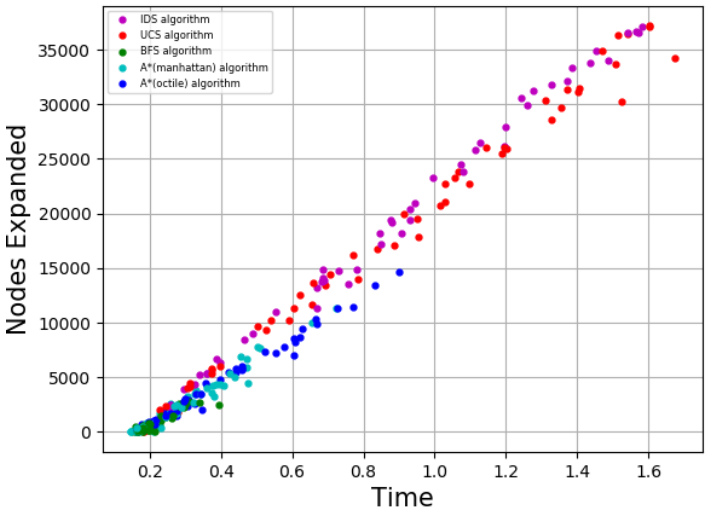
Por exemplo se movo 5 vezes na direção x e 8 vezes na direção y, seriam 5 movimentos diagonais e 3 movimentos laterais; o custo seria (5*1.5 + 3*1) = 10.5 = 8*1 + 5*0.5 = menor custo.

Experimentos

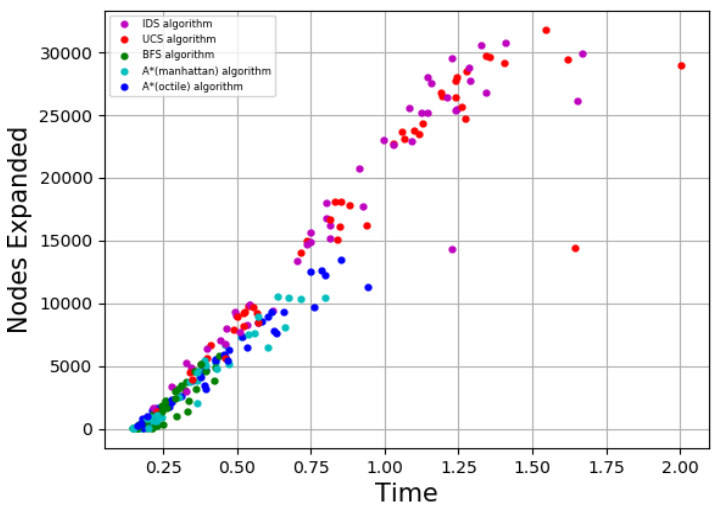
Foram gerados pares de estados final e inicial e cada par foi testado (Tests.py) nos quatro algoritmos propostos. Os testes foram realizados nos três mapas disponibilizados. Nestes pares, foram feitas médias aritméticas dos parâmetros e disponibilizado na Tabela 1.

Seguem os resultados:

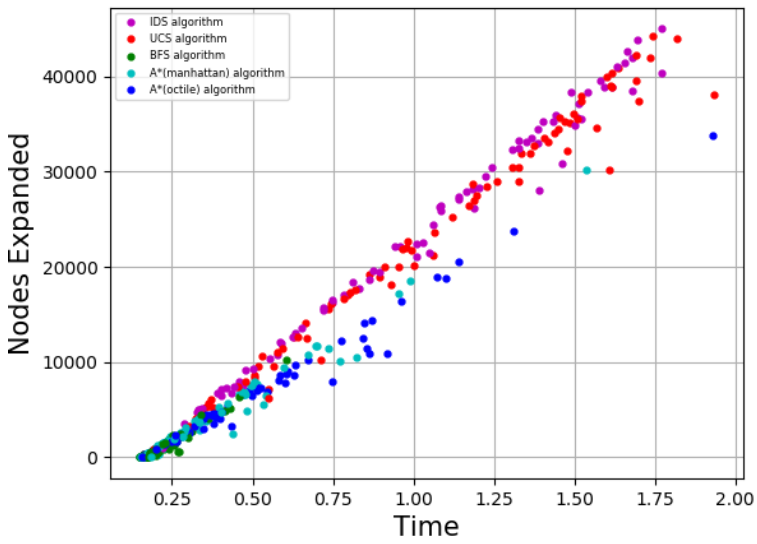
map1.map



map2.map



map3.map



		Testes de execução			
		Média do tempo d execução	Média do número de nós expandidos	Média da diferença entre a solução encontrada e a solução ótima	
mapa1.map	IDS	0.873	18856	28.5	Experimentos realizados: 52
	UCS	0.908	18853	0	
	BFS	0.214	839	30.3	
	A* Manhattan	0.305	2808	1.6	
	A* Octile	0.387	4363	0	
mapa2.map	IDS	0.867	17285	26.2	Experimentos realizados: 69
	UCS	0.916	17596	0	
	BFS	0.254	1619	39.8	
	A* Manhattan	0.340	3401	0.1	
	A* Octile	0.421	4909	0	
mapa3.map	IDS	1,000	22849	29.5	Experimentos realizados: 83
	UCS	1,036	22851	0	
	BFS	0.227	1264	54.7	
	A* Manhattan	0.352	3958	0.5	
	A* Octile	0.451	5662	0	

Tabela 1: Média aritmética dos parâmetros de execução após

realizado os testes nós quatro algoritmos.

Pelos experimentos realizados, podemos inferir algumas informações, e prós e contras de cada algoritmo.

IDS: Dentre os estudados, é considerado o mais ingênuo, pois não utiliza de nenhuma informação para encontrar um resultado, não é ótimo, expande uma quantidade elevada de nós e a solução encontrada não se aproxima muito bem da solução ótima.

UCS: Encontra sempre uma solução ótima, porém também expande uma quantidade elevada de nós.

BFS: Expande o menor número de nós e é o mais rápido entre todos, porém, é um algoritmo guloso e a solução encontrada na maioria dos casos é a pior aproximação da solução ótima dentre os algoritmos estudados.

A Manhattan*: Na maioria dos casos para este problema, o algoritmo encontra uma solução ótima, em poucos casos é encontrado um caminho sub ótimo, ou seja, a heurística aproxima muito bem a solução ótima expandindo um baixo número de nós, e o tempo de execução é um pouco pior que o *BFS*, ficando em segundo lugar.

A Octile*: Quando usamos uma heurística consistente e admissível como a *Octile,* este é considerado o melhor algoritmo para o problema proposto, pois seu tempo de execução fica em terceiro lugar, dentre os estudados, e ele sempre retorna uma solução ótima expandido um baixo número de nós.