Relatório do Trabalho Prático 01 - Introdução à Inteligência Artificial

Thiago Martin Poppe [1]

[1]Matrícula: 2017014324

Abstract. *Report requested to evaluate the practical work 01 for the discipline of Introduction to Artificial Intelligence (DCC642 - TM), from the Federal University of Minas Gerais (UFMG).*

Resumo. *Relatório solicitado para avaliação do trabalho prático 01 da disciplina de Introdução à Inteligência Artificial (DCC642 - TM), da Universidade Federal de Minas Gerais (UFMG).*

1. Introdução

O trabalho prático 01 consiste em implementar diversos algoritmos de busca com e sem informação para auxiliar o personagem Pac-Man em sua busca por comidas em uma fase. Todos os algoritmos foram implementados sobre um código base disponibilizado pela UC Berkeley.

Algoritmos de busca com e sem informação são muito importantes para a área de inteligência artificial servindo de base para diversas aplicações do mundo moderno, como o clássico exemplo de navegação através de mapas como o Waze.

Dentre os algoritmos de busca com e sem informação, tivemos que implementar:

1. Busca em profundidade (DFS): Algoritmo de busca sem informação que irá explorar o estado mais profundo da árvore, ou grafo, de busca. Para sua implementação, foi utilizada uma estrutura de dados chamada de Stack, ou Pilha. Ao armazenarmos os estados a serem explorados nessa estrutura de dados, teremos uma ordem de First-In Last-Out, que reflete essa exploração em profundidade.

2. Busca em largura (BFS): Algoritmo de busca sem informação que irá explorar os estados em ordem First-In First-Out, podendo ser modelada da mesma forma que uma DFS, porém trocando a estrutura de dados para uma Queue, ou Fila. A BFS, ao contrário da DFS, sempre irá encontrar uma solução ótima nos labirintos do Pac-Man, assumindo que o custo de locomoção é 1 em qualquer direção.

3. Busca de Custo Uniforme: O algoritmo de busca de custo uniforme, ou algoritmo de Dijkstra, se trata de uma busca sem informação que faz uso do paradigma guloso. Nele, sempre iremos considerar o custo do caminho feito até então, explorando sempre o estado com menor custo associado. Para termos um melhor desempenho, armazenaremos os estados em uma fila de prioridades, sendo a prioridade o custo do caminho feito, obtendo assim o próximo estado a ser explorado com custo $O(1)$. Essa busca sempre irá encontrar uma solução ótima desde que os custos de locomoção não sejam negativos.

4. Busca gulosa: Algoritmo de busca com informação que utiliza uma heurística para encontrar a resposta do problema. Similar ao algoritmo de Dijkstra, usaremos uma fila de prioridades, porém, nesse caso, a prioridade será apenas o valor retornado pela heurística, podendo ser por exemplo a distância euclidiana do estado atual com o goal. Diferentemente da busca de custo uniforme, a busca gulosa não é ótima.

5. Algoritmo A*: Algoritmo de busca com informação que, de forma geral, é uma mescla entre a busca de custo uniforme e a busca gulosa. Dessa vez, a prioridade a ser armazenada na fila de prioridades será $f(n) + h(n)$, onde $f(n)$ representa o custo do caminho percorrido e $h(n)$ o valor retornado pela heurística. Aplicando uma heurística admissível e consistente, juntamente com o custo do caminho percorrido, torna o A* um algoritmo de busca poderoso que sempre encontrará uma solução ótima. O uso da heurística melhora principalmente o tempo gasto para computar essa solução, visto que a heurística, por sua vez, direciona a busca, explorando assim menos nós que a busca de custo uniforme, por exemplo.

Para a implementação dos algoritmos de busca foi utilizado estruturas auxiliares já existentes no código, como Stack, Queue e PriorityQueue. Os 5 algoritmos foram implementados de forma similar, apenas mudando as sutilezas apresentadas acima. Como foi solicitado que retornássemos uma lista de ações, representando a solução, utilizei uma tupla para armazenar tanto o estado atual quanto as ações feitas até então, facilitando assim o retorno da resposta ao explorarmos o goal.

2. Metodologia e Análise Experimental

Foi escolhido as informações relativas à pontuação do Pac-Man, tempo, custo e número de estados expadindos para encontrar o caminho retornado para realizar as comparações entre os algoritmos. As fases escolhidas para computar tais métricas foram: **bigMaze, contoursMaze, mediumMaze, openMaze, bigCorners, greedySearch, mediumCorners, mediumDottedMaze, smallSafeSearch, testClassic, tinySearch** e **trickySearch**, que possuem uma ou mais comidas à serem coletadas pelo jogador. Essas fases foram escolhidas por possuírem uma dificuldade mediana, fazendo com que a análise experimental não demorasse muito. Vale ressaltar que a heurística **nullHeuristic** não foi utilizada durante as comparações visto que ao utilizarmos ela, o algoritmo de busca gulosa degenera para uma BFS, dado que o custo de locomoção sempre será 1, e o A* degenera para uma busca de custo uniforme.

2.1. Resultados para algumas fases

2.1.1. Fase contoursMaze

Na tabela a seguir, temos uma comparação feita entre os algoritmos na fase **contoursMaze**, que nada mais é do que uma fase vazia com o jogador posicionado no meio e que possui uma comida no canto inferior esquerdo.

	SearchAgent	Level	Score	Cost	Time	NodesExpanded	ProblemType	Heuristic
7	DFS	contoursMaze	425	85	0.0	85	PositionSearchProblem	None
8	BFS	contoursMaze	497	13	0.0	170	PositionSearchProblem	None
9	UCS	contoursMaze	497	13	0.0	170	PositionSearchProblem	None
10	GS	contoursMaze	497	13	0.0	13	PositionSearchProblem	euclideanHeuristic
11	GS	contoursMaze	497	13	0.0	13	PositionSearchProblem	manhattanHeuristic
12	A*	contoursMaze	497	13	0.0	60	PositionSearchProblem	euclideanHeuristic
13	A*	contoursMaze	497	13	0.0	49	PositionSearchProblem	manhattanHeuristic

Por ser uma fase extremamente fácil, os algoritmos não levaram muito tempo para encontrar um caminho até o objetivo. Porém, note que os algoritmos que utilizam alguma heurística sempre exploram menos nós do que aqueles que não utilizam, onde no caso da busca gulosa, os estados expandidos foram justamente o caminho encontrado pelo algoritmo.

2.1.2. Fase bigMaze

Analisando a tabela, percebemos que a DFS foi o melhor algoritmo para a fase **bigMaze**, possuindo a maior pontuação possível e o menor número de estados expandidos. Isso se deve ao fato dessa fase possuir apenas 1 rota que leva o Pac-Man até o seu objetivo. Sendo assim, um algoritmo simples e rápido como a DFS consegue se sair melhor do que a busca de custo uniforme, por exemplo.

Mesmo que todos os algoritmos encontrem a solução ótima, a escolha de uma boa heurística tem um papel fundamental para reduzir o número de estados explorados pelo algoritmo. Ao utilizarmos a distância de manhattan, sempre teremos um número menor ou igual de estados expandidos do que se usarmos a distância euclidiana. Isso se deve ao fato de que a distância de manhattan modela uma versão menos relaxada do problema, que desconsidera as paredes e permite apenas movimentos na horizontal e vertical, similar à movimentação do personagem; enquanto que a distância euclidiana relaxa mais o problema, permitindo, além disso, movimentos na diagonal.

	SearchAgent	Level	Score	Cost	Time	NodesExpanded	ProblemType	Heuristic
0	DFS	bigMaze	300	210	0.0	390	PositionSearchProblem	None
1	BFS	bigMaze	300	210	0.0	620	PositionSearchProblem	None
2	UCS	bigMaze	300	210	0.2	620	PositionSearchProblem	None
3	GS	bigMaze	300	210	0.0	471	PositionSearchProblem	euclideanHeuristic
4	GS	bigMaze	300	210	0.0	466	PositionSearchProblem	manhattanHeuristic
5	A*	bigMaze	300	210	0.1	557	PositionSearchProblem	euclideanHeuristic
6	A*	bigMaze	300	210	0.2	549	PositionSearchProblem	manhattanHeuristic

2.1.3. Fase trickySearch

Para concluir, a tabela a seguir representa as comparações feitas para a fase **trickySearch**, que possui um tamanho relativamente pequeno, com mais de uma comida e múltiplos

caminhos possíveis para encontrar uma solução válida. Podemos observar que de fato os únicos algoritmos ótimos são a BFS, UCS e A*. Porém, note que a busca gulosa obteve um resultado relativamente próximo do ótimo, expandido muito menos estados.

	SearchAgent	Level	Score	Cost	Time	NodesExpanded	ProblemType	Heuristic
35	DFS	trickySearch	414	216	0.0	361	FoodSearchProblem	None
36	BFS	trickySearch	570	60	2.1	16688	FoodSearchProblem	None
37	UCS	trickySearch	570	60	10.6	16688	FoodSearchProblem	None
38	GS	trickySearch	529	101	2.3	6848	FoodSearchProblem	foodHeuristic
39	A*	trickySearch	570	60	4.5	9551	FoodSearchProblem	foodHeuristic

A heurística utilizada para a busca gulosa e A* foi uma heurística personalizada, denominada de **foodHeuristic**. Essa heurística foi pensada com o objetivo de diminuir o número de estados explorados pelo A* para menos de 10 mil estados, nessa fase em específico. A heurística implementada consegue realizar essa proeza, fazendo com que o A* expanda 9551 estados ao invés de 16688 como a BFS e a busca de custo uniforme.

Para que o A* retorne a solução ótima, a heurística deve ser tanto **admissível** quanto **consistente**. Para facilitar o processo de encontrar uma boa heurística, optei por partir de uma que simplesmente calcula a distância de manhattan entre o Pac-Man e seu objetivo, já que ela possui esses dois aspectos devido à desigualdade triangular respeitada pela métrica de distância. Em outras palavras, $d(x, z) \leq d(x, y) + d(y, z)$, sendo d a função de distância e y um nó intermediário entre x e z.

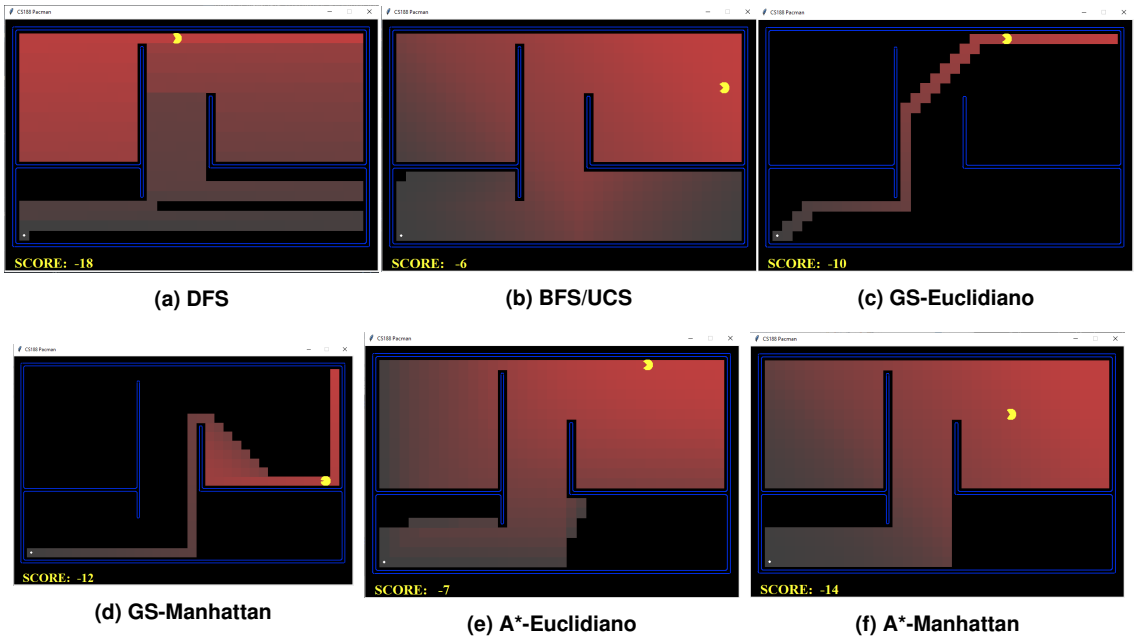
A partir desse ponto, tentei manipular as distâncias computadas de diversas formas, como, por exemplo, somando elas, o que gerava um resultado excelente. Porém, ao analisarmos com mais detalhes, percebemos que essa heurística não está correta, visto que a estimativa retornada pode ser maior do que o custo real a ser percorrido pelo Pac-Man até o seu objetivo, ou seja, o custo de capturar todas as comidas da fase.

Após outras abordagens, a escolhida foi uma heurística simples, que retorna como estimativa o máximo das distâncias entre o Pac-Man e cada uma das comidas. Essa heurística é tanto **admissível** e **consistente** por natureza devido ao uso da distância de manhattan, definida como $d(v, w) = |v_x - w_x| + |v_y - w_y|$.

De modo mais formal, a heurística será **admissível** visto que $h(n) \leq h^*(n)$, sendo $h^*(n)$ o custo real para alcançar o objetivo a partir de um estado n, devido ao fato de que para coletarmos todas as comidas do mapa devemos no mínimo percorrer a distância calculada até a comida mais distante, o que necessariamente deve ser menor ou igual ao custo real de coletar todas as comidas. E, a heurística será **consistente** pois para cada estado n, todo sucessor n' gerado por uma ação a sobre n obrigatoriamente irá seguir a desigualdade triangular, em outras palavras $h(n) \leq cost(n, a, n') + h(n')$, novamente por conta de estimarmos um valor através de uma métrica de distância que respeita essa desigualdade.

2.1.4. Fase openMaze

As métricas calculadas para a fase **openMaze** não foram tão diferentes se compararmos com as de outras fases, porém, essa fase tem um formato muito interessante que permite com que a gente analise a forma como cada algoritmo realiza a busca pelos estados. Os estados com coloração mais forte são os que foram explorados primeiro, e os com coloração mais fraca por último.



(a) DFS (b) BFS/UCS (c) GS-Euclidiano

(d) GS-Manhattan (e) A*-Euclidiano (f) A*-Manhattan

Figura 1. Ordem de exploração dos estados

A partir das imagens [1], conseguimos encontrar um certo padrão para cada algoritmo de busca:

- DFS: realiza uma varredura na vertical indo da esquerda para a direita.

- BFS/UCS: tanto a BFS quanto a UCS realizaram a busca em forma de "camada", explorando quase todos os estados do labirinto. Os dois algoritmos foram equivalentes pois o custo associado com as ações para se locomover sempre é 1, sendo assim, a UCS degenera para uma BFS.

- GS: tanto para a heurística que utiliza a distância euclidiana quanto para a que utiliza a distância de manhattan, o algoritmo foi capaz de explorar poucos estados até encontrar o caminho para o objetivo, devido ao direcionamento fornecido pelas heurísticas.

- A*: o comportamento desse algoritmo foi bem similar a execução da BFS, porém, percebemos a forte influência do direcionamento da heurística quando o Pac-Man se direciona para a parte debaixo do labirinto.

2.2. Resultados gerais

A seguir, teremos duas tabelas que mostram a média das métricas para cada algoritmo nas fases que possuem apenas 1 comida (PositionSearchProblem) e nas fases que possuem

múltiplas comidas (FoodSearchProblem).

SearchAgent	Heuristic	Score	Cost	Time	NodesExpanded
A*	euclideanHeuristic	423.75	86.25	0.050	348.25
	manhattanHeuristic	423.75	86.25	0.100	338.50
BFS	None	423.75	86.25	0.000	435.25
DFS	None	329.25	180.75	0.000	299.25
GS	euclideanHeuristic	402.75	107.25	0.000	174.25
	manhattanHeuristic	418.75	91.25	0.000	161.50
UCS	None	423.75	86.25	0.075	435.25

(a) PositionSearchProblem

SearchAgent	Heuristic	Score	Cost	Time	NodesExpanded
A*	foodHeuristic	535.625	63.125	1.7500	2387.50
BFS	None	535.625	63.125	1.3500	4839.75
DFS	None	153.250	445.500	0.1750	545.75
GS	foodHeuristic	505.000	93.750	1.4375	2289.75
UCS	None	535.625	63.125	3.2875	4839.75

(b) FoodSearchProblem

Figura 2. Média das métricas

A partir das tabelas [2], podemos concluir que de fato os únicos algoritmos ótimos são a BFS, UCS e A*, dado uma heurística admissível e consistente. Além disso, percebemos que em média, os algoritmos de busca gulosa e DFS expandem bem menos estados que os demais, e que, de fato, a heurística criada (foodHeuristic) consegue reduzir drasticamente em média o número de estados expandidos se compararmos com uma heurística nula, como por exemplo o algoritmo de busca de custo uniforme.

3. Conclusão

Com o uso do código fornecido pela UC Berkeley, foi possível, de forma bem criativa e interessante, implementar e estudar os diversos algoritmos de busca, com e sem informação, que formam o alicerce da inteligência artificial. Por mais que o código seja extenso e um pouco confuso, ele é bem documentado, e com atenção e paciência é possível entender o que devemos fazer.

A implementação dos algoritmos não foi tão difícil como eu imaginava, sendo a maior dificuldade realmente o entendimento do código. Fora isso, as buscas são bem similares, mudando apenas a estrutura de dados utilizada para armazenar os estados e o uso ou não de uma heurística.

A maior dificuldade encontrada foi durante a elaboração das análises de comparação dos algoritmos, principalmente devido ao caso degenerado onde a busca de custo uniforme sempre se comporta de forma igual a busca em largura. Seria uma ótima ideia poder modificar o custo de locomoção para deixar mais evidente a diferença entre esses dois algoritmos, principalmente dentro do contexto onde os custos não são os mesmos, o que faz com que a BFS deixe de ser um algoritmo ótimo.

4. Bibliografia

Foi utilizado apenas os slides e vídeos da disciplina [Chaimowicz] como material de consulta e estudo.

Referências

Chaimowicz, L. Slides vídeos da disciplina de introdução à inteligência artificial.