Report Content

Name: Jie Liu
StudentID: jil251

1. First, briefly describe your project setup: which version of Python? Basic computer configurations:

Version of Python	python 2.7
Software	Pycharm / Spider
Model Name	MacBook Pro
Processor Name	Intel Core i5
Processor Speed	2.3 GHz
Total Number of Cores	Total Number of Cores
Memory	16 GB

2. If you discussed the project with someone, list their names and briefly describe what you talked about. Relatedly, if you consulted any sources outside of the class notes and the textbook, please give citations to them here.

First one I discussed with: Singh Ravneet, we discussed about whether each search algorithm uses the explored list or not. Our result is, for graph search, use explored list; for tree search, no explored list.

	Algorithms	Graph/tree search
Uniformed search	BFS	Graph
	DFS	Tree
	unicost	Graph
	IDDFS	Tree
Informed search	Greedy	Tree
	A*	Graph
	IDA*	Tree

Second one I discussed with: Jianfeng He and Yue Dai, we discussed about the existing loop problem in the search graph and the consuming time of then pancake puzzle.

3. Report any known bugs you have.

One important bug is: “TypeError: “object' is not iterable” I faced this bug when I used the FILO queue and Priority Queue to store the object. I find both FILO queue and Priority Queue are not implemented by a list, Priority Queue is implemented by a heap, so there is no iterator for it. To solve this bug for FILO queue, I used a deque instead of FILO queue, for Priority Queue, I rewrite an iterable version for it.

4. For each puzzle type, enumerate your action expansion order:

I list one action expansion order for each puzzle, the rest actions for other search algorithms can be found in transcript file. One action expansion order for jugs puzzle on`test_jug.config` searched by BFS is:

```
(0, 4 (3, 1)(3, 0)(0, 3)(3, 3)(2, 4)(2, 0)(0, 2)(3, 2)(1, 4)(1, 0)
For greedy, h(n) is euclidean distance, the action order is:
(0, 0)(3, 0)(0, 3)(3, 3)(0, 4)(3, 1)(0, 1)(1, 0)
For greedy, h(n) is manhattan distance, the action order is:
(0, 0)(3, 0)(0, 3)(0, 4)(3, 1)(0, 1)(1, 0)
For greedy, h(n) is dot product, the action order is:
(0, 0)(0, 4)(3, 0)(0, 3)(3, 4)(3, 1)(0, 1)(1, 0)
For greedy, h(n) is cosine similarity, the action order is:
(0, 0)(0, 4)(3, 4)(3, 1)(0, 1)(1, 0)
```

Algorithm	h(n)	Maxfrontier	Generatednodes	Explored	Execute time
BFS	None	9	42	11	0.000969886779785
DFS	None	10	42	11	0.000762939453125
Greedy	euclidean	7	26	7	0.000761985778809
Greedy	manhattan	6	22	6	0.000574111938477
Greedy	dot product	4	24	7	0.00066089630127
Greedy	cosine similarity	4	16	5	0.000874996185303

One action expansion order for jugs puzzle on `test_cities.config` searched by unicast is below. Action list is:

```
C00,C01,C02,C03,C04,C10,C11,C12,C13,C14,C20,C21,C22,C23,C24,C30,C31,C
32,C33,C34,C40,C41,C42,C43,C44
searched by greedy, h(n) is euclidean distance :
Action list is as below:
C00,C11,C22,C33,C44
searched by greedy, h(n) is manhattan distance :
Action list is as below:
C00,C11,C22,C33,C44
searched by astar, h(n) is manhattan:
Action list is as below:
C00,C11,C22,C33,C44
searched by astar, h(n) is euclidean:
Action list is as below:
C00,C11,C22,C33,C44
```

Algorithm	h(n)	Maxfrontier	Generatednodes	Explored	Execute time
unicost	None	49	141	24	0.0194399356842
greedy	euclidean	21	27	4	0.00401401519775
greedy	manhattan	21	27	4	0.004065990448
Astar	euclidean	21	27	4	0.00401782989502
Astar	manhattan	21	27	4	0.00408101081848

For the pancake puzzle, the data in the test_pancakes1.config and test_pancakes2.config are too long to finish them within 30 minutes, so I first make the length of pancake begin with 5 to 11, and check when the search algorithms can get the goal state. searched by astar, h(n) is euclidean distance :

Algorithm	H(n)	Pancake length	Maxfrontier	Generatednodes	Explored	Execute time
Astar	euclidean	5	376	770	154	0.0481340885162
Astar	euclidean	6	8815	24636	4106	3.57055592537
Astar	euclidean	7	95837	293489	41927	933.947599173
Astar	euclidean	8	None	None	None	beyond 30 minutes
Astar	euclidean	9	None	None	None	beyond 30 minutes
Astar	manhattan	5	249	460	92	0.0268158912659
Astar	manhattan	6	379	630	105	0.0377058982849
Astar	manhattan	7	4585	7917	1131	0.687799930573
Astar	manhattan	8	38847	79912	9989	27.5603380203
Astar	manhattan	9	None	None	None	None

5. Give the name of the heuristic function (so that the grader can supply it as an argument in the command line). If you made up any heuristic function(s) in addition to the ones we specified, explain what it is and why you designed it that way.

I designed four heuristic functions in the project, Euclidean distance, Manhattan distance, dot product, and cosine similarity.

The Euclidean distance is specified in the project, when you supply it as an argument in the command line, you just need to type: euclidean.

Manhattan distance can be used to measure the similarity between two data points, in the project, the jugs water's state, the location of cities, and the state of the pancakes can be thought as the vector, if the distance is long, we can think their similarity is low, otherwise, the similarity will be high. When you supply it as an argument in the command line, you just need to type: manhattan.

Dot product can be used to measure the similarity between two data points, in the project, the jugs water's state, the location of cities, and the state of the pancakes can be thought as the vector. Algebraically, the dot product is the sum of the products of the corresponding entries of the two sequences of numbers. Geometrically, it is the product of the Euclidean magnitudes of the two vectors and the cosine of the angle between them. When you supply it as an argument in the command line, you just need to type: dotproduct.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. In the product, some vectors are zero

vectors, so I additionally add a constant to the product of the two vector modes. When you supply it as an argument in the command line, you just need to type: cosinesimilarity.

6. show a transcript of the expected run.

This is indicated in the transcript log file.

7. Additional discussion points: Here, you may want to run some test cases on more search strategies than those required for the transcript. Address the following issues:
For each of the three puzzle types, what do you think is the best search strategy, and why? You should take all four factors into considerations (completeness, optimality, time complexity, space complexity).

For water jug puzzle, based on the experiments, greedy search algorithm with appropriate designed heuristic function performs best. BFS is optimal and complete compare to DFS, but the drawbacks of BFS is time complexity, it has exponential time complexity, while the DFS only has linear time complexity. And they have the same time complexity, because the search space is not large for the jug puzzle problem, it means the time complexity is not a big concern. While the BFS can find an optimal goal state compare to the DFS, so BFS outperforms DFS in the water jug puzzle.

For path puzzle, based on the experiments, Astar performs among the three algorithms if an appropriate heuristic function is designed. The greedy search also has a good performance, but it is not optimal and complete. For the uniform cost, although it is optimal and complete, but it has an exponential time and space complexity, the result is the unicast has the longest executional time.

For the pancake puzzle, an appropriate heuristic function plays an important role during the searching process. It indicates the initial state is also very important, if we can set a proper initial state, we can receive the result soon, while we cannot arrive at the goal state under some situations. Thus, choose proper initial state and heuristic function is very import when the search space increases exponentially. Based on the experiments, the space is a big concern in this puzzle, because the search space is exponentially large, hence, the search algorithms those have large space time complexity should not be considered, like BFS, Uniform cost etc. For the heuristic function, Manhattan distance out performs Euclidean distance when the search algorithm if Astar.

For the water jugs problem and the burnt pancakes problem, is your heuristic admissible and consistent? Explain.

There are 4 heuristic functions in the project, including the Euclidean distance, water jugs problem and the burnt pancakes problem, the cost from parent to child node is 1, so all the four heuristic function are not admissible, the condition for the admissible is:

$$h(n) \leq h_{perfect}$$

This condition can't be satisfied all the time. Based on the definition of the consistency:

$$h(n) \leq cost(a, n, n') + h(n')$$

The four designed heuristic functions are also not consistent.

8. You may also include additional discussions on any observations that you find surprising and/or interesting.

Observations:
1. For the water jug problem, because the search space is not too large, the advantage of the DFS, depth limited DFS, iterative deepening DFS on space complexity is not that apparent.
2. An appropriate heuristic function for the informed search algorithms are very import, it can reduce the time consuming.
3. The initial state for the pancake puzzle is extremely important, the goal state can be reached under proper initial state, while a bad initial state will make the algorithm execute for a long time and can't receive the final result.

The reasons why astar is hard to get the result

4. The astar can be thought as a combination of unicost search and greedy search, if the cost by uicost is very small, the performance of astar is similar to greedy; while if the cost by greedy is very small, the performance of astar is similar to unicost. So I think if we set a small value to the heuristic function $h(n)$, or to $g(n)$, like 0.0001; we can get the result of the sstar on the pancake puzzle.