BBM405 - Fundamentals of Artificial Intelligence
Assignment 1 Report

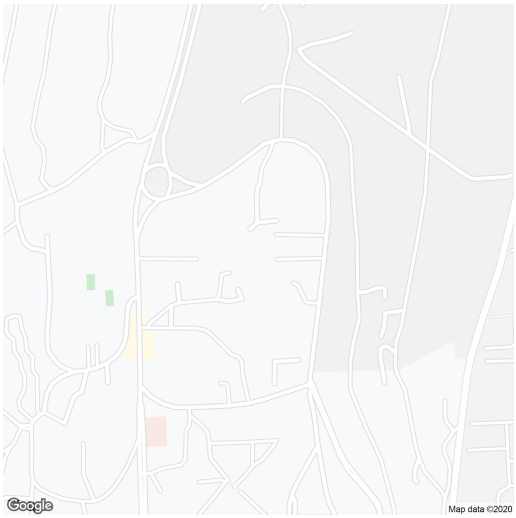
Name & Surname	: Ali Yunus Emre Özköse
ID	: 21527271
Course	: BBM405 - Fundamentals of Artificial Intelligence
Advisor	: Pınar Duygulu Şahin
Due	: 17.05.20
E-mail	: b21527271@cs.hacettepe.edu.tr

Contents

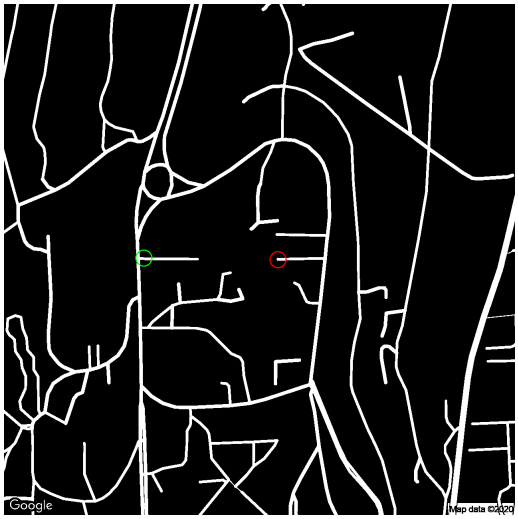
1 Introduction

Primary objective of the assigmnent is applying different search algoritms to different search problems and analyzing outcomes. The assignment includes 2 different problem which are applied 4 different search algorithm for each problem. Problems are explained in their own sections broadly. Hence, I refer to section 2.1 and section 3.1. Applied search algoritms are depth first search(DFS), breadh first search(BFS), uniform cost search(UCS) and A* search(A*). Mentioned search algoritm implementations are got from [1] which the supervisor shared in Piazza.

2 Problem 1



(a) 1280x1280 image of bird's eye view of Hacettepe University. Image is retrieved with Google Static Maps API



(b) 1280x1280 image of road map of Hacettepe University. Green circle is start point (ATMs and red circle is goal point (Computer Science Department))

Figure 1: Problem: Finding a convenient map between given two points.

2.1 Problem Review

Moving operation from a point to another is a challenging problem in robotics. There are a number of issues that have to be considered such as mapping&localization (visual or normal slam), navigation, etc.. One of them is path findinding. Path finding includes finding the shortest path which helps the robot to change position itself from one point to another one. For analyzing, 2 point are choosen for goal/start states and fixed during the experiment. However, points can be changed to any white point in road map.

To be more concreate for path finding problem, Hacettepe University is choosen as an experiment environment. Figure 1 shows two different version of bird's eye view of Hacettepe University.

[1] https://github.com/chitholian/AI-Search-Algorithms

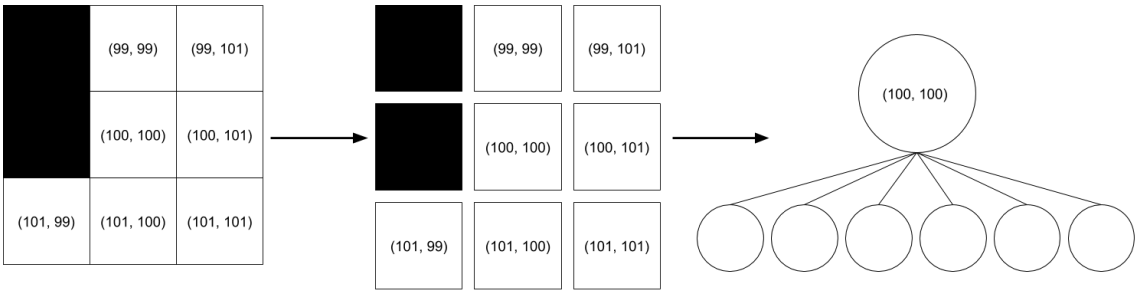


Figure 2: Creating graph example: Adding edges of pixel coordinates(100, 100) in road map

Figure 1a is retrieved with Google Static Maps API[2]. The objective is reaching a goal pixel from a start pixel. Figure 1b shows road map of Hacettepe University.

Road map is retrieved from Figure 1a with a basic color thresholding which is shown in Algorithm 1. Threshold is adjusted by experimentally. Then the image is converted to graph by following shared implementation (mentioned in section 1).

Data: 1280x1280 image array *map*
Result: Binary image which shows only roads
threshold = 250;
indices = map>threshold;
map[indices] = 255;
map[!indices] = 0;

Algorithm 1: Color Thresholding ro create road map from bird's eye view

The graph of this problem has relatively more nodes and edges than the other one. Related statistics are given in Table 1. Nodes consists of all white pixels of the road map image. Edges consists of adjacent white pixel connections. Black pixels are not consired as node as can be shown in Figure 2

	1280x1280 image	3840x3840 image
Number of node	156.735	1.410.615
Number of edge	1.136.200	10.893.814

Table 1: Statistics of the road map graph

Path planning is well-studied challenge. Hence, there are a lot of proposed heuristic function such as manhattan distance, euclidean distance, etc.. For this experiment, euclidean distance is used. The equation is given below:

[2]https://developers.google.com/maps/documentation/maps-static/intro

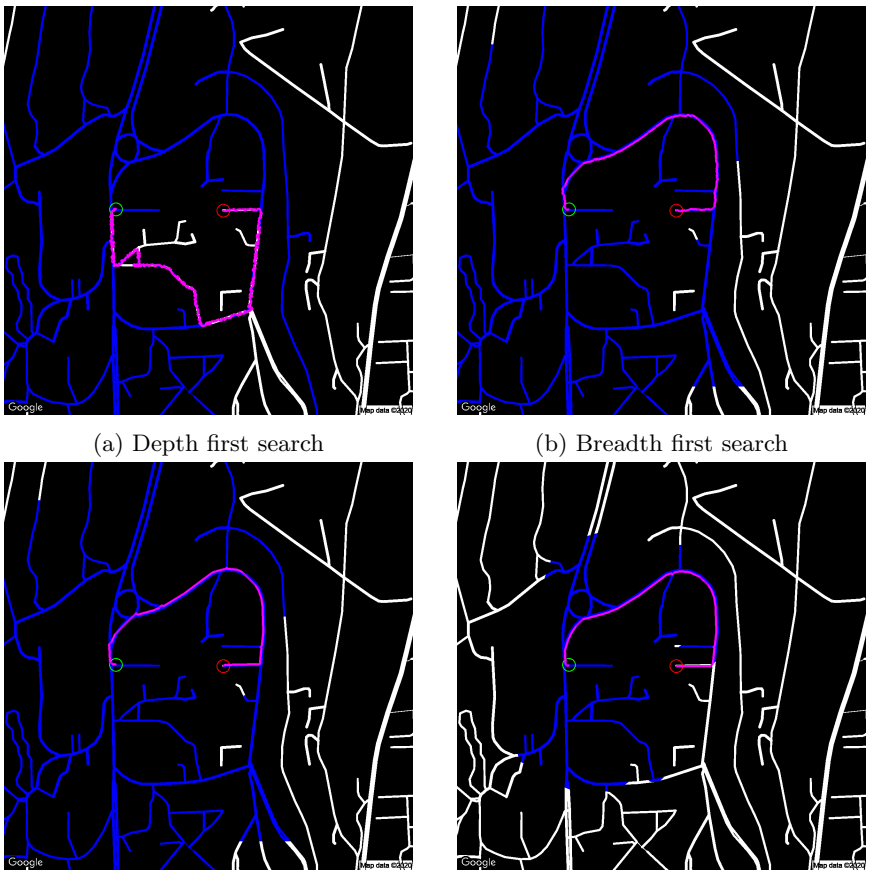
$$d(p,q) = \sqrt{\sum_{i=1}^{2} (q_i - p_i)^2} \tag{1}$$

where p and q stand for coordinates of start and goal pixels.

2.2 Results & Analysis

2.2.1 Qualitative analysis

Convenient paths which are found by each search algoritms are shown in Figure 3. It is very clear that the least number of visited node is extracted by A* search. If figues are looked in detail, the most clear (the shortest) path is created by A* search. Following analysis are made by considering Figure 3.



(a) Depth first search

(b) Breadth first search

(c) Uniform cost search

(d) A* search

Figure 3: Search result path is drawn on the given map for each search algorithm. Green and red points are start and goal states respectively. Pink path stands for traced path from start pixel to goal pixel.

Completeness All algoritms are able to find at least one solution. Hence, we can say that all algoritms are complete.

Optimality DFS is clearly far away from optimal solution. However, remaining 3 algorithm are very close to each other. If we see in detail, BFS's path has a bumpy line. It is not exactly straight from start pixel to goal pixel. UCS has a nearly straight line, but has slight deviations on the path. If these 2 paths which are created by BFS and UCS are compared with A*, it is clear to see that UCS and BFS are not optimal. A* produce the best/optimal path in this experiment.

2.2.2 Quantitative analysis

Time Complexity Since created graph is explicit data structure, we can say that time complexity is $|V| + |E|$ where $|V|$ is number of white pixels and $|E|$ is number of edges. However elapsed times are also observed to make comparision between each algorithm. Statistics can be found in Table 2. It is interesting that A* can find optimal solution for this experiment, but it takes longer compared to BFS and DFS which can not find optimal solution. DFS and BFS implementations do not include some heuristic function. Hence A* takes a little bit long time. In addition, I though if the map would be higher size (three times: 3840x3840), will elapsed time and number of visited nodes will be changed or not. Table 2 also shows that results. It is parallel to previous map size.

Method Name	1280x1280 image		3840x3840 image	
	#of visited node	Elapsed Time (s)	#of visited node	Elapsed Time (s)
DFS	107.706	0.181650	609.981	1.057504
BFS	100.464	0.174751	895.202	1.823937
UCS	100.467	0.489716	895.187	5.188453
A*	37.254	0.238551	314.495	2.358068

Table 2

3 Problem 2

3.1 Problem Review

Because of creativity and originality requirements, I want to create a unique problem and formulate in a different way. COVID-19 Open Research Dataset Challenge (CORD-19)[3] is a call for AI researchers to contribute COVID-19 literature. The Challenge includes a great dataset which named CORD-19 dataset. The dataset consists of academic papers from several archives such as PMC, BioRxiv/MedRxiv, Arxiv, etc..

First step is creating a unique problem from this dataset. Since dataset contains a lot of papers (*63,000 scholarly articles, including over 51,000 with full text*), searching related papers is really challenging. The idea is that if 2 papers are defined as start and goal paper as a node in a paper graph, a search algorithm can find goal paper from start paper as well as related papers in the path. As a result, we have 2 related papers and also a related paper list.

To create a graph from the dataset, papers are considered as nodes and title similarities between each papers are used to make connections between 2 papers. Document similarities are calculated with Spacy[4] which is a natural language processing library for Python. Specifically, *en_core_web_sm* model is used for similarity. Steps are also shown in Algorithm 2.

Data: *papers*
Result: Created graph by using title and abstract similarities for each pair of paper
edge_list = [];
edge_threshold = 0.75;
for *for each pair of papers* **do**
  paper1, paper2 = pair_of_papers;
  title_similarity = find_similarity(paper1.title, paper2.title);
  abstract_similarity = find_similarity(paper1.abstract, paper2.abstract);
  if *title_similarity > edge_threshold* **then**
    edge_list.add(Edge(paper1, paper2));
  end
end

Algorithm 2: I used document similarity to make connections between each paper which is a node in the graph.

Finding a convenient heuristic function is also critical issue for A* search. If a paper would be more related with goal paper than another paper, the path should include this paper instead other one. Hence, the heuristic function **h()** should maximize relation during searching. Heuristic function h() is defined as below:

[3]https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
[4]https://spacy.io/usage/vectors-similarity

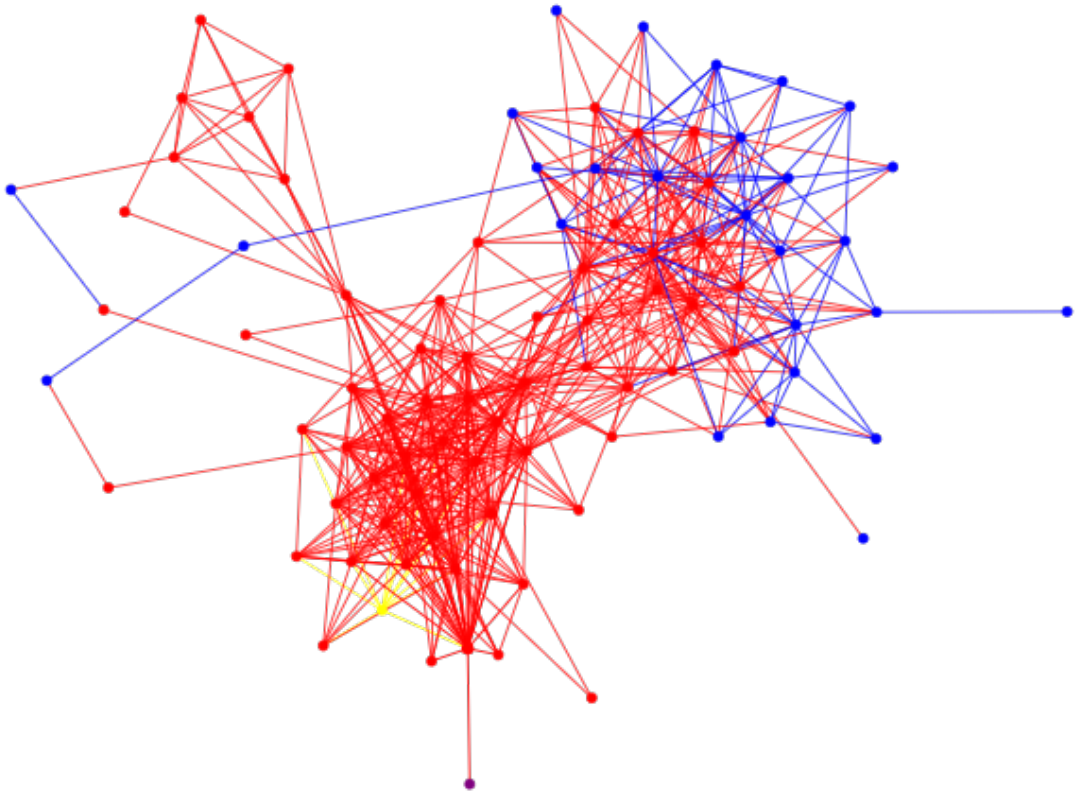


Figure 4: Nodes represents papers and edges are stands for strong paper similarities.

$$h(paper1.abstract, paper2.abstact) = \\ 1 - spacy.document\_similarity(paper1.abstract, paper2.abstract) \quad (2)$$

The objective of this function is maximizing similarities between current papers and goal paper. In shared implementation, heuristic was designed for minimizing distance. Contradiction occurs here. Hence a distance score is calculated from similarities and used for heuristic value. Spacy similarity function returns a value between 0 and 1 which stands for percentage of similarity. The similarity is substracted from 1.

Since calculating similarites takes a long time (several hours), similarites are calculated once, and experiments are done by using pre-calculated similarites. Although CORD-19 dataset is a huge dataset, 500 papers are used in experiments, because of cost of similarity calculation. For visualization of the created graph, 100 papers are visualized as a graph in Figure 4. The Figure also shows a path which is created by A* search algorithm. Yellow node is start node, purple node is goal node. Red colors stands for visited nodes. Results are shown in Figure 6 and Figure 5. Graph is created with blue nodes initially. Yellow node is assigned as start node. Purple node is assigned as goal node. Red nodes stands for visited nodes.

	500 papers	100 papers
Number of node	500	100
Number of edge	19.991	631

Table 3: Statistics of paper graphs

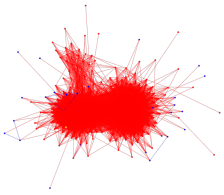
3.2 Results & Analysis

3.2.1 Qualitative analysis

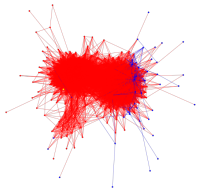
Completeness All algorithm are able to find a path. It means that at least one solution can be found in the graph, and all of them are complete.

Optimality If Figure 5 and Figure 6 are looked in detail, we can see that DFS searchs almost every node to find goal state. It is easy to deduce that DFS is not optimal. The same observation can be made for BFS and UCS. However A* does not look every node in the graph.

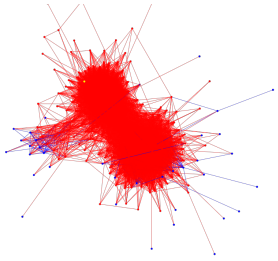
Let's consider the graph of 100 papers. It is not easily seen in Figures, but there is a short path between goal and start where this path can be shown in Figure 4. Hence we can say that A* is not optimal in this experiment. Another reason of this situation can be heuristic function. Current heuristic function is similarity of abstracts of papers. Better heuristic function can lead an optimal search algorithm. It can be done by considering body similarities instead of body of papers. However I can not conduct an experiment with this setup because of cost of power.



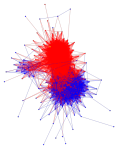
(a) Depth first search



(b) Breadth first search

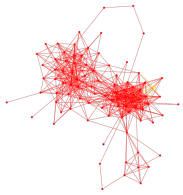


(c) Uniform cost search

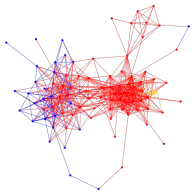


(d) A* search

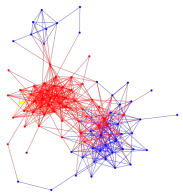
Figure 5: Visualization of paper graph with 500 papers. Only orientations are different in graphs.



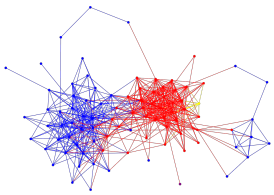
(a) Depth first search



(b) Breadth first search



(c) Uniform cost search



(d) A* search

Figure 6: Visualization of paper graph with 100 papers. Only orientations are different in graphs.

3.2.2 Quantitative analysis

Time Complexity Elapsed times and number of visited node statistics are written in Table 4. As can be predicted in qualitative analysis, A* visits the least number of node during search. It shows quality of heuristic function. Heuristic function works well where it can maximize similarity correctly. At least, it is enough well to make A* a good algorithm compared to other search algorithms. Heuristic function can be replaced with body of papers' similarities, but it takes a long time to calculate them. Hence, experiment with body of papers could not be done.

Method Name	500 papers		100 papers	
	#of visited node	Elapsed Time (s)	#of visited node	Elapsed Time (s)
DFS	453	0.00067	90	0.00024
BFS	440	0.00344	73	0.00009
UCS	448	0.00667	51	0.00016
A*	306	0.00504	42	0.00015

Table 4