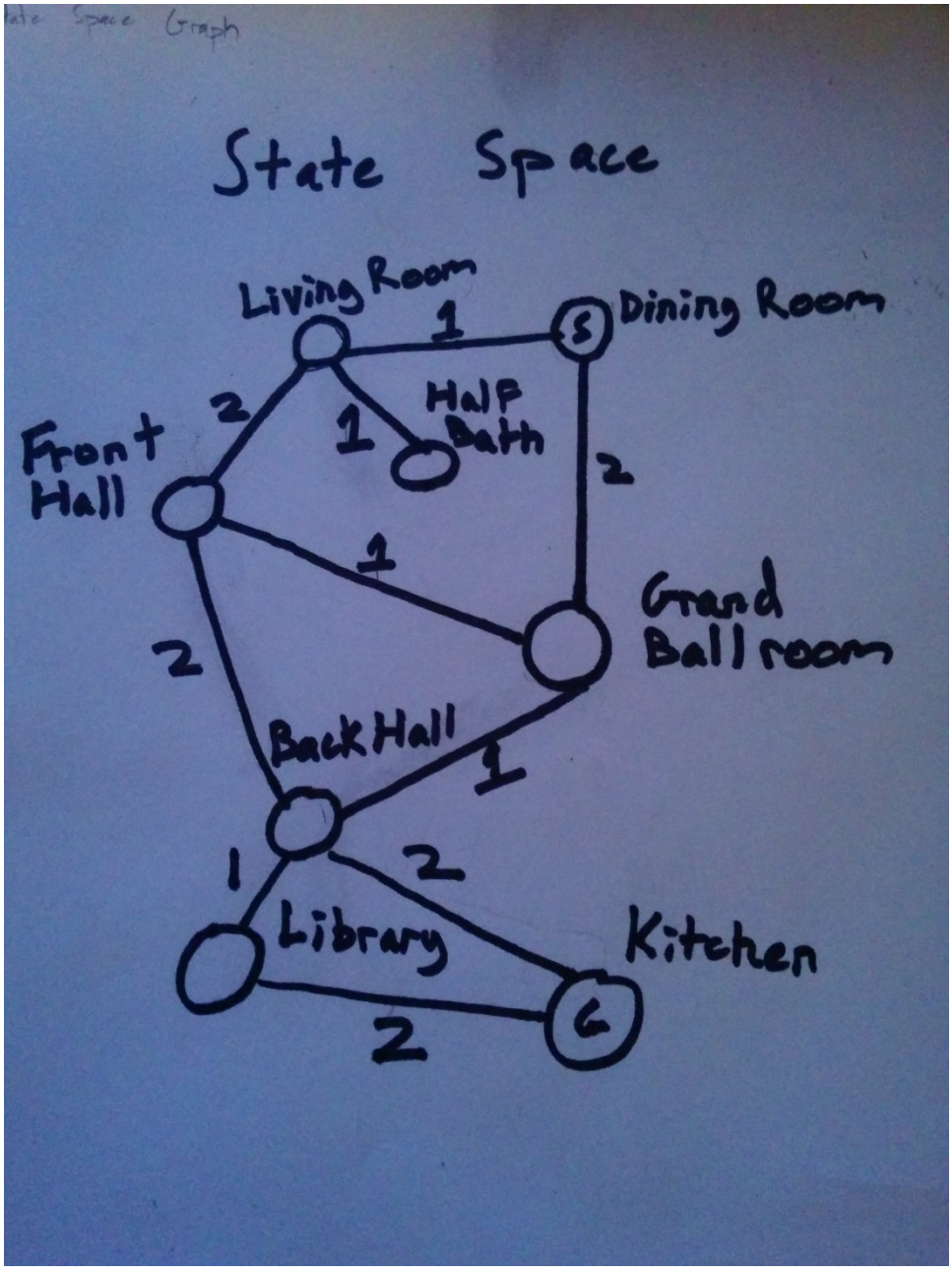
Ian Wilkes

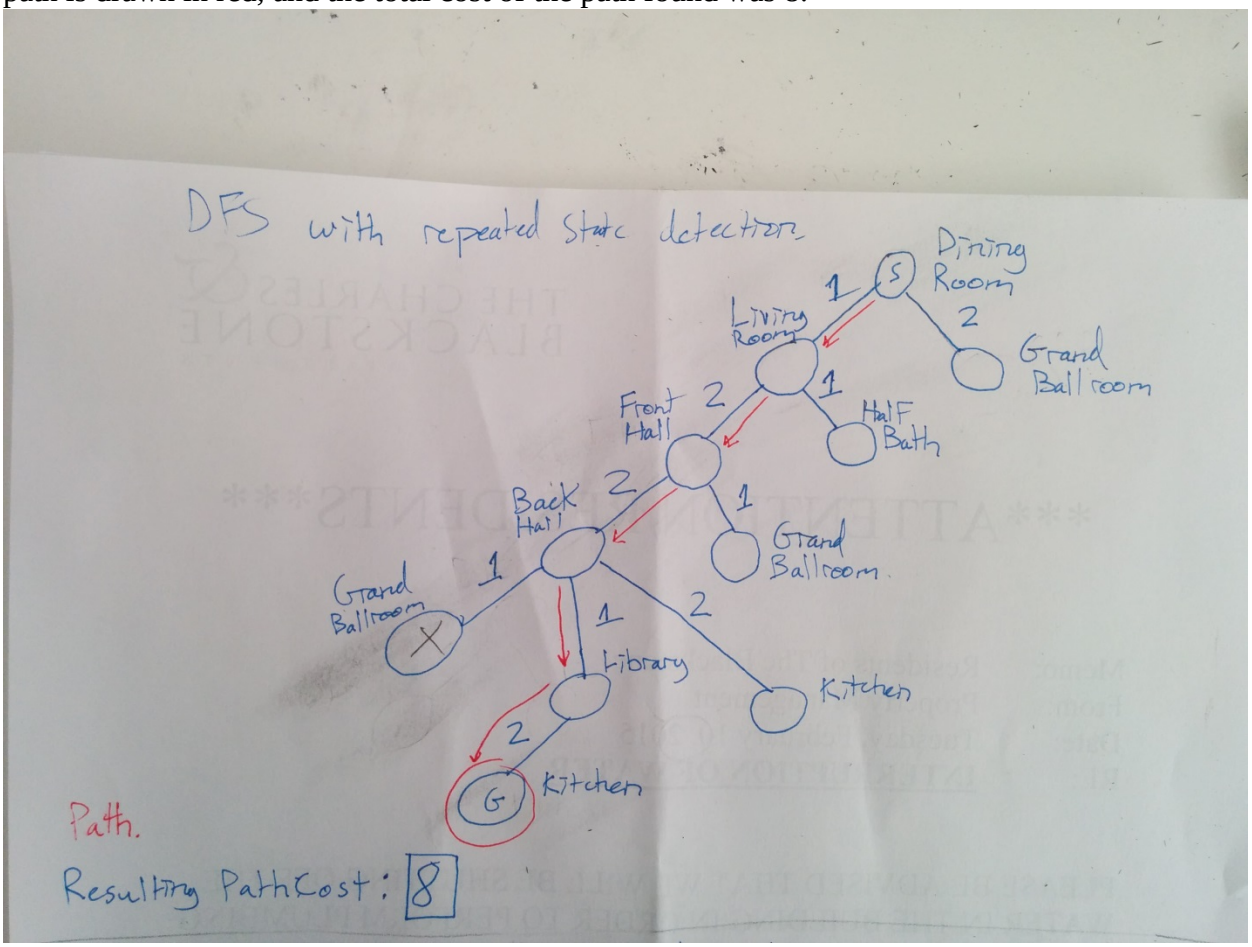
600.335 Artificial Intelligence Spring 2015

Homework 1: Search

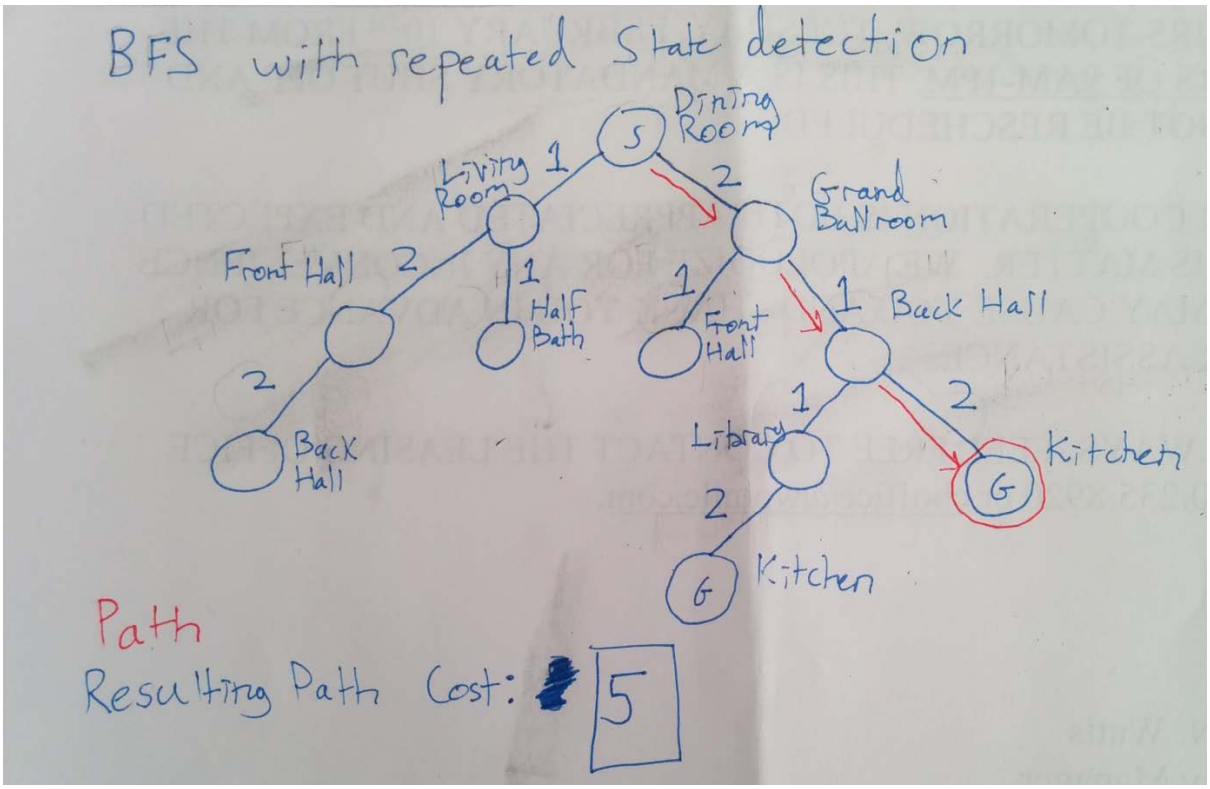
1. The following image contains the state-space for the robot navigation problem.



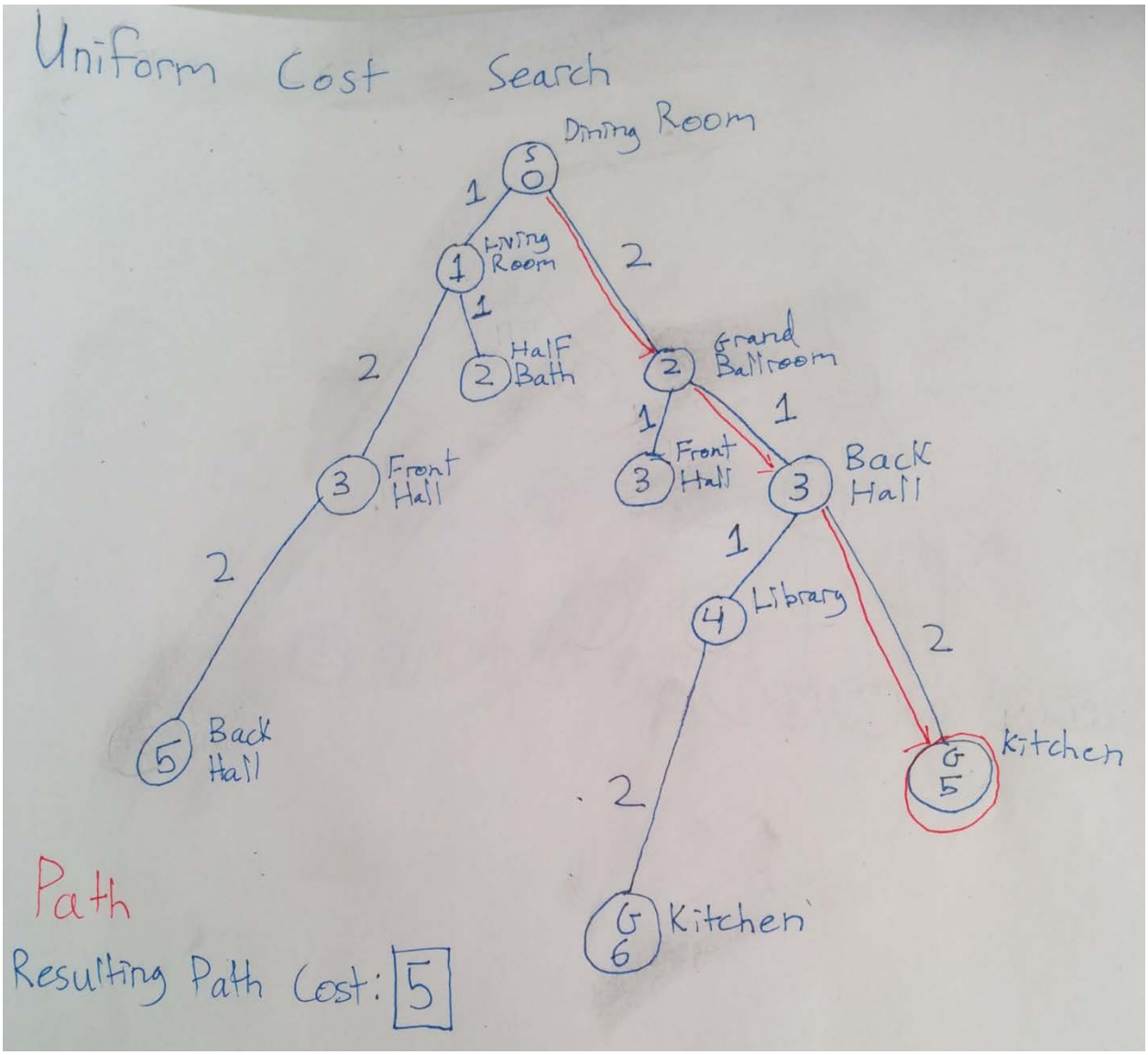
2. The following image contains the DFS search tree. For my DFS, I have included nodes which would have been added to the stack of search nodes, but I have also used repeated state detection, which skips nodes which have already been visited, and does not add nodes which have been previously visited to the search stack. Nodes are considered to be visited when they come out of the search stack and their children are expanded. Also, I have assumed that the first door the robot scans is the first one which it will explore. The path is drawn in red, and the total cost of the path found was 8.



3. The Following image contains the BFS search tree I have created. I have included nodes which would be added to the search queue, but only new nodes which have not been visited previously. I count a node as visited only when it has come out of the search queue and its children are expanded. So when the robot expands nodes, it only includes those which have not previously been visited. Also, if the robot tries to visit a node which has already been visited, it does not revisit that node or expand its children, as this has already been done elsewhere on the search tree. The path used is drawn in red on the image. The total cost of that path was 5.



4. The following is an image of a uniform cost search tree. This uniform cost search had the same repeated state detection as the two previous searches. Nodes with the same cost were visited in the order in which they were added to the priority queue. The path found is drawn in red, and the total cost of that path is 5.


Uniform Cost Search
Dining Room
S
0
1
Living Room
1
2
2
Half Bath
2
Grand Ballroom
2
1
1
3
Front Hall
3
Front Hall
3
Back Hall
2
1
4
Library
2
5
Back Hall
2
G
5
Kitchen
2
G
6
Kitchen
Path
Resulting Path Cost: 5

5. The results below are for the performance of the BFS, DFS and A* algorithms I have implemented. For all three I have included code to prevent nodes from being visited multiple times. A node is considered to be visited if it comes out of the search data structure. If a node has already been visited, it is not expanded again, and if a visited node comes out of the search data structure to be visited again, it is skipped.

map number	BFS:			DFS:			A*:		
	nodes	time (s)	cost	Nodes	time (s)	cost	nodes	time (s)	cost
1.	8	0.049	7	8	0.052	7	8	0.090	7
2.	15	0.052	14	15	0.051	14	15	0.053	14
3.	34	0.051	14	21	0.050	14	28	0.049	14
4.	45	0.056	22	44	0.063	26	45	0.082	22
5.	64	0.051	15	64	0.095	21	56	0.094	15
6.	1844449	1.417	9499	1230805	1.709	610433	1283490	1.431	9499
7.	39	0.050	∞	39	0.093	∞	39	0.106	∞
8.	25	0.055	21	18	0.062	23	20	0.048	15

6. For my A* heuristic I used the sum of the Manhattan and Euclidean distances all divided by two. Both are individually admissible heuristics, as they always estimate at most the actual distance to the goal, and never more, so by summing the two heuristics, the result can never be more than twice the actual distance to the goal. Thus, dividing by two, we again have an admissible heuristic. This heuristic also dominates the Euclidean distance, as Manhattan distance is always greater than or equal to the Euclidean distance. I chose this instead of regular Manhattan distance as by this method, there are fewer nodes with exactly the same heuristic value than would be the case for regular Manhattan distance, thus making this new heuristic more useable. My version of A* either matched the path cost of both the BFS and DFS searches in the latter maps, or provided better paths. In almost all cases, the A* algorithm expanded many fewer nodes than BFS, and most of the time fewer than the DFS, except on a few occasions. On the latter maps, the running time of A* is approximately comparable with that of DFS, and a little bit slower than DFS.

7. A gridworld type problem is an example where DFS might fail but BFS could succeed. Due to the large numbers of cycles encountered in gridworlds, DFS, which allows for nodes to be expanded multiple times, can get stuck when it encounters a cycle, because the DFS may simply explore one branch of the tree which loops back on itself at some point. BFS will not encounter this same problem, because it explores all branches of the search tree, so if a goal is reachable, BFS will eventually reach it, and while cycles will add to the number of nodes to explore, BFS will always explore other nodes as well.

8. Basic DFS is much more memory efficient than DFS, so on large graphs where the branching factor for the search tree is large, DFS may be able to find a solution when BFS may run out of memory, due to the exponential nature of the memory usage of BFS. This is because BFS stores all of the possible paths to all the nodes being explored, while for DFS, all the paths to be explored come off of the same branch of the search tree, so fewer nodes must be stored. Thus, the example of a problem where BFS might fail would be the n-queens problem for large values of n. Due to the extremely large

branching factors, the memory usage of BFS would be astronomical, while DFS would only need to store a few nodes at a time, and since there are no cycles, a fixed solution depth, and multiple solution states, DFS will be able to complete even in cases where BFS would run out of memory.

9. A* will perform equally to BFS in gridworld cases where the goal node cannot be reached. Both search strategies will eventually visit all nodes reachable from the start state and due to the presence of cycles in gridworlds will simply run forever back and forth in the connected section of the graph. Neither search strategy will terminate, so we can say that they both perform equally. Eventually each will run out of available memory, and as BFS and A* store roughly the same amount of data, they will run out at approximately the same time.

10.

$h(n) = 2\ for\ all\ n$ This heuristic is not admissible, because if the goal is one space away from the start, then the actual cost to reach the goal is 1, so the heuristic overestimates.

$h(n) = 0\ for\ the\ goal, 1\ otherwise$ This heuristic is admissible, because the value is always less than or equal to the total distance to the goal, but it is not useful, because all non-goal nodes will have the same value, which defeats the purpose of the heuristic. This will result in A* performing almost exactly the same as uniform cost search.

$h(n) = Euclidean\ distnce\ from\ current\ node\ to\ goal\ node$ This is a fairly good heuristic, because it is admissible, as Euclidean distance is always less than or equal to the total path length to a goal in a gridworld. It is also fairly useful, as nearly all spaces in the gridworld will have distinct values in $h(n)$ thus making the priority queue used in A* more useful at distinguishing better paths.

$h(n) = Twice\ the\ Euclidean\ distance\ from\ the\ current\ node\ to\ the\ goal\ node$ This heuristic is not admissible, as in the case of map1 from the project. The Euclidean distance from the start to the goal is 7, but twice that is more than the actual distance to the goal, so this is not an admissible heuristic.

$h(n) = Manhattan\ distance\ from\ current\ node\ to\ goal\ node$ This heuristic is admissible, as in a gridworld, the minimum path length is always dictated by the Manhattan distance to the goal. It is fairly useful, as it will always predict the total distance to reach the goal from the current node, but because there are many configurations which can produce the same Manhattan distance, it may not be extremely useful, as many nodes will be expanded because they will be equally good in the priority queue of A*.

$h(n) = One\ half\ the\ Manhattan\ distance\ from\ current\ node\ to\ goal\ node$ This heuristic is also admissible, as the Manhattan distance is equal to the path length to the goal, and half that distance is always less than or equal to that distance. This heuristic runs into the same problem as the Manhattan distance, but is slightly worse, because it is dominated by the Manhattan distance, so less emphasis is placed on nodes being close to the goal, and more is placed on simply being farther away from the start node in the A* search. This search will almost certainly result in more nodes being expanded than for the regular Manhattan distance

heuristic, because the underestimation of the remaining distance to the goal will cause other extraneous nodes to be visited before finally visiting the goal node, because their heuristic costs will be artificially low.

11. The Euclidean distance between two locations on the globe may take aircraft through said globe. This is somewhat of a problem when trying to fly. A better heuristic could be the shortest arc length on the surface of the earth from the origin to the destination.