Grad AI. 15-780 Fall, 2006

Homework 1

- *Homework deadline: 10:30am on Oct 4*
- *Please download Matlab helper code and data files at class website*
- *Please print your code and hand it in with the hard copy of your homework. Also send a copy of your code by e-mail to both TAs.*

1. **Search**. This section will involve comparing the performance of DFS, BFS, DFID, and A* on a set of mazes.

   (a) Figure 1 shows a simple maze. The thick lines between cells represent the walls. At each location in the maze you can move in at most one of four directions (constrained by the walls). You are not able to revisit a cell if it has already been visited before. You consider these directions in the following order:

   $$N \rightarrow E \rightarrow S \rightarrow W$$

   Find a shortest path from a to x. Suppose the start is the first thing pushed on the stack. Show the expansion order for

   - DFS, BFS
   - Heuristic search using h = horizontal number of cells to the goal
   - A* using $f = g + h$

   Is this heuristic h admissible? Is there any other admissible heuristic h' we could use? Please provide at lease another one and explain which is a "better" heuristic. (Note that heuristic h' is said to be more informed ("better") than h if both are admissible, $h'(n) \geq h(n)$ for every non-goal n, and $h'(n') > h(n')$ for at least one non-goal n').

   (b) When the maze becomes larger, we use a single Boolean matrix representing whether it is possible to move from a given cell in each of the four directions. Each row in the matrix corresponds to exactly one grid space in the maze and each column corresponds to one direction. For example, the table below shows the entries for cells a, b, and c in Figure 1. The legal moves from the cell a are East and South.

	N	E	S	W
a	0	1	1	0
b	1	0	1	0
c	1	1	1	0

   The maze is indexed in column order from North to South, then West to East. The maze in a 4x6 grid as in Figure 1 would have indices as shown in the table

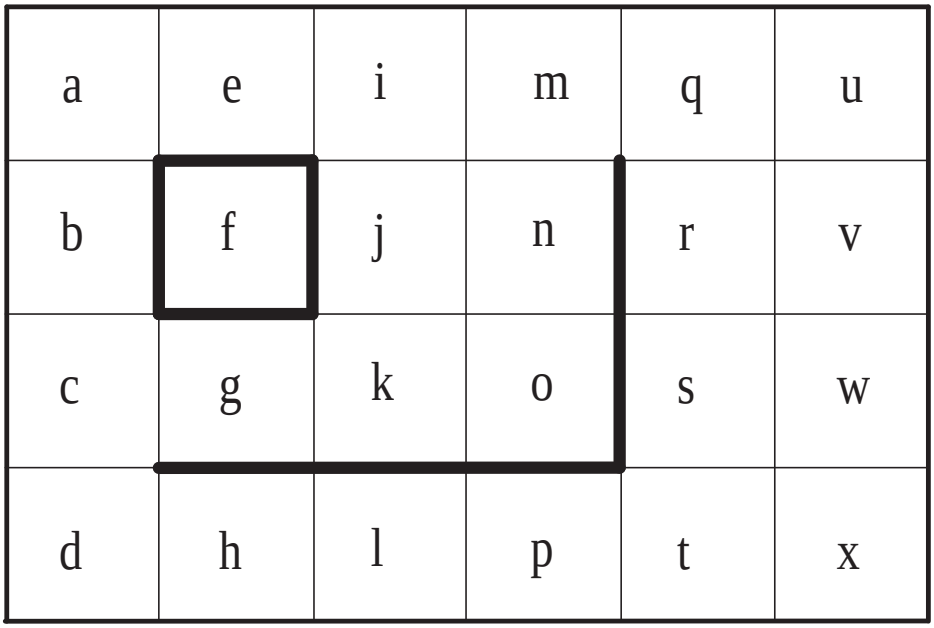

Figure 1: A simple maze. The thick lines between cells represent the walls.

below, where North toward the top of the page. The start location is always the top left cell (index 1) and the goal is always the bottom right cell (index 24 in the table below). Please use script `load_maze` to load the maze file. You can display the loaded maze with function `draw_maze`, with or without index label.

1	5	9	13	17	21
2	6	10	14	18	22
3	7	11	15	19	23
4	8	12	16	20	24

Movement between cells in the maze is accomplished by directly calculating the successor cell's index. Specifically, we can define the movements on an $Y \times X$ maze as (consider the index order for the maze) :

- Move North: $ind = ind - 1$
- Move South: $ind = ind + 1$
- Move East : $ind = ind + Y$
- Move West : $ind = ind - Y$

with the appropriate checks on the bounds and validity of movements. Two helper functions are provided to map between (X, Y) coordinate and the matrix index:

- `maze_XY_from_index`: Gets the X and Y coordinates of the location at index.
- `maze_index_from_XY`: Gets the index from the X and Y coordinates.

Randomness is introduced in this task in terms of breaking ties uniformly at random. For this part of the problem please ignore the expansion order of $N \rightarrow$

$E \rightarrow S \rightarrow W$ described in the previous part, and have BFS and DFS push in a random order.

Please program your solutions in Matlab. Helper files are provided. If you feel strongly about using another language, please contact one of the TAs.

In your programming, search nodes should be opaque data structures, with operations `get_start`, `get_neighbors` and `test_goal`. This will facilitate using the same search functions in Problem 2. Write four functions: `BFS`, `DFS`, `DFID`, and `ASTAR` (A*). These functions should produce both the path found and the number of cells expanded. The maze file corresponding to Figure 1(`maze0.txt`) and also other five maze files are available at the website. You can start with the simple maze and then run your search functions on each maze multiple times (20-30) with your `BFS`, `DFS`, `DFID`, but run `ASTAR` (A*) only once. Specifically, for each maze record the following statistics for `BFS`, `DFS`, and `DFID`:

- The solution found.
- The average number of cells expanded.
- The largest number of cells expanded.
- The smallest number of cells expanded.

For A*, just submit the solution and the number of cells expanded. To help you to implement A*, a priority queue data structure is provided to start from. The functions you want to use are:

- `pq_init`: Initialize a priority queue.
- `pq_set`: Reset the priority of an element, or insert it if it's not already there.
- `pq_pop`: Remove and return the first element. (The first element is the one with smallest numerical priority value). It does not return a random node of lowest priority.

Please submit the above observations and all codes. To prevent the computer hanging on a large map, restrict DFS to expand at most 10,000 cells.

2. **Constraint Satisfaction Search**. This problem is a variation of the missionaries and cannibals problem from the Russel and Norvig textbook Chap 3. A group of people must cross a river using a small raft. Unfortunately not everyone gets along and there are certain rules that must be followed in order to get everyone across safely. The group consists of a woman and two girls, a man and two boys, and a policeman with a thief. If you leave certain people alone with others, trouble will ensue. For example, the thief will only behave if the policeman is on the same bank.

   The complete rules are as follows:
   (`http://jayisgames.com/archives/2006/06/raft_iq_puzzle.php`)

   - A maximum of two people can be on the raft at a time.
   - One adult must be on the raft to operate it.
   - The man cannot be with any of the girls without the woman present.

- Conversely, the woman can't stay with the boys without the man there.
- The thief must be with the policeman or be alone.

There is a flash game online at the above web site. You can get a sense of the problem from playing. (The game is in Chinese, but the only text is on the opening screen. Click the big circle to start playing.)

(a) Formulate this problem precisely, making only those distinctions necessary to ensure a valid solution and giving states, operators, start, and goal of the problem.

(b) Write Matlab code to describe the problem and write another data structure for search nodes (apparently different from Problem 1), including a new `get_neighbors` function. Run your `DFS`, and `ASTAR` from the first problem. Record the number of nodes expanded and the runtime of each algorithm. Make sure they can solve this problem correctly. Submit all the codes and the solution you get using your problem formulation.

3. **Spatial Planning**. This section will involve problems related to configuration space and robot kinematic motion planning.

(a) Figure 2 shows a work space with three obstacles. The mobile robot can only translate in this space (2 dof). Please draw approximately the c-space when the robot has the shape:

- a circle with radius 5
- a 5×5 square

Note the scale information is shown in Figure 2 using little grid cells. When you draw the c-space please indicate with a line for each robot shape the outer most point that the center of the robot can possibly occupy.

(b) Consider a robot arm with two sequential links (Figure 3). Each link is a straight-line segment of length 1. The arm is installed on a fixed base. Joint J_1, which locates at (0, -1.2), has one degree of freedom and rotates in $[-\pi/2, \pi/2]$. Joint J_2 also has one degree of freedom and rotates in $[0, \pi]$. q_1 and q_2 are the planar angles of link L_1 and L_2 respectively. There are three circular obstacles in the workspace, with parameters listed in the table below:

center	radius
(-1, 0.5)	1.0
(1.6,-1.1)	0.4
(1.5, -3)	0.5

The script `run` initializes the graphical display (Figure 4). You can play with the graphical arm by moving the sliders corresponding to the joint angles to get a sense of the robot arm. The script `run` uses a function `forward` that computes the forward kinematics of the two-link-arm. That is it take the joint angles as inputs and compute the end effector position in (x, y) as well as the positions of all the joints.

Figure 2: A work space with three obstacles. (included in the data files: cspace/obstacles.PNG)

Please code a loop over all configurations in Matlab at the resolution $\Delta q = 0.001\pi$ rad and test for intersections (Do not consider the width of the arms). Print the two-dimensional c-space as a $q_1 - q_2$ plot. Submit your code and the plot.

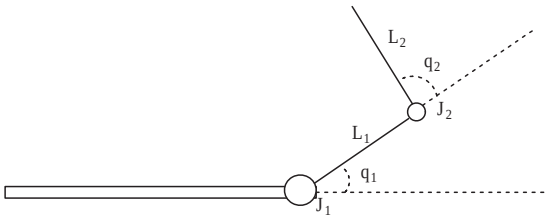

Figure 3: A two-link-arm illustration.

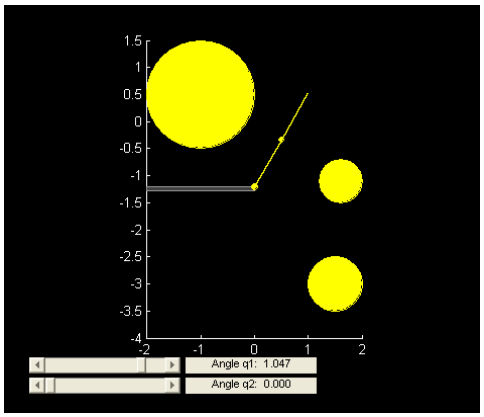

Figure 4: A graphical arm animation.