Grad AI. 15-780 Fall, 2007

Homework 1

- *Homework deadline: 10:30am on October 4*
- *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 (gholling@andrew.cmu.edu and thlin@cs.cmu.edu).*

1. **Propositional Logic and Satisfiability (40 pts)**.

   (a) Write the following sentences in propositional logic. Define your own literals, but be consistent throughout all sentences. (2 pts each)

      i. Mary went to the mall, Alice went to the mall, and either Teddy did not go to the mall or Leroy went to the mall.

      ii. If Teddy went to the mall or Mary did not go to the mall then Alice did not go to the mall.

      iii. Leroy did not go to the mall and Alice went to the mall only if Teddy and Mary did not both go to the mall.

      iv. If Leroy went to the mall and Mary went to the mall then either Alice and Teddy did not both go to the mall or Alice went to the mall and Teddy did not go to the mall.

   (b) Convert your sentences in Part 1(a):i-iv to conjunctive normal form. (2 pts each)

   (c) (From Russell and Norvig) Decide whether each of the following sentences is valid, unsatisfiable, or neither. Verify your decisions using truth tables or equivalence rules of Fig. 7.11 in Russell and Norvig. (2 pts each)

      i. $(Smoke \Rightarrow Fire) \Rightarrow (\neg Smoke \Rightarrow \neg Fire)$

      ii. $((Smoke \wedge Heat) \Rightarrow Fire) \Leftrightarrow ((Smoke \Rightarrow Fire) \vee (Heat \Rightarrow Fire))$

      iii. $(Smoke \Rightarrow Fire) \Rightarrow ((Smoke \vee Heat) \Rightarrow Fire)$

      iv. $Big \vee Dumb \vee (Big \Rightarrow Dumb)$

   (d) Prove the following using the inference rules on pages 211-214 of Russell and Norvig. You can also Wikipedia "Category:Rules of inference" for some better descriptions. HINT: You WILL need to use Modus Ponens, Modus Tollens, and DeMorgan. (3 pts each)

      i. Given:
      $P \Rightarrow S$
      $E \vee \neg S$
      $\neg(\neg P \vee N)$
      Prove:
      E

ii. Given:
$A \vee B$
$C \Rightarrow D$
$A \Rightarrow E$
$\neg(E \vee D)$
Prove:
$C \vee B$

(e) (Adapted from Russell and Norvig). Write the following English sentences as propositional logic sentences. Again, you may choose your own literals, but be consistent. (10 pts total)

> If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

Given your sentences, can you prove that the unicorn is mythical? How about magical? Horned? If you can prove it, do so using the inference rules of propositional logic.

2. **Configuration Spaces and RRTs (20 pts)**

(a) Figure 2 shows a work space with three obstacles. The mobile robot can only translate (not rotate) in this space (2 dof). Please draw (approximately, by hand) the c-space when the robot has each of the following shapes: (3 pts each)

i. A 1 cm $\times$ 1 cm square.
ii. A circle of radius 5 cm.
iii. A 5 cm $\times$ 1 cm rectangle (top-down length is 5 cm).

The little grid cells in Figure 2 represent 1cm each. You can print out a copy of the figure and draw directly on it, if you wish.

(b) (You may wish to complete Problem 3 before answering this question). The space in Figure 1 can pose problems for RRTs. **Explain why.** To get around this problem, you might think of using a search method like A*. **What would be the advantages and disadvantages of A* over RRTs? How could you get around some of the disadvantages of using A*?** Give your comparison in terms of optimality, completeness, and efficiency (both in memory and in time). Note: the space is continuous when initially given. For concreteness, you may assume that you are given a list of the coordinates of the corner points of each obstacle and of the workspace as a whole. (11 pts)

3. **PROGRAMMING PROBLEM: Search (40 pts)**

Please e-mail your code for this section in a zip file to both TAs. To avoid mixup, please e-mail with the following subject line and archive file name:

Subject: 15-780 Homework 1 Submission
Archive name: (yourID)-hw1.zip
Where (yourID) is your andrew or cs ID.

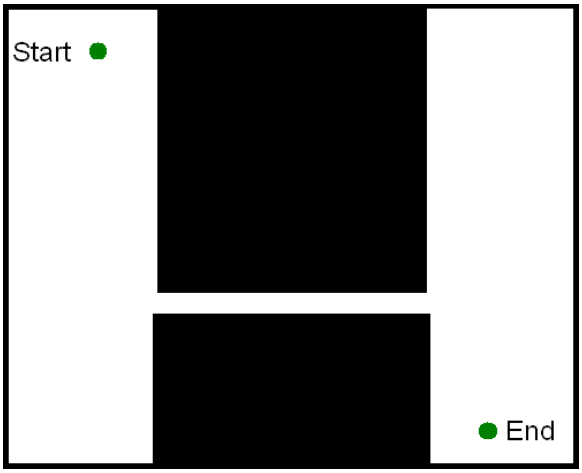

Figure 1: Continuous environment path planning problem that RRTs may have difficulty solving.

WARNING: This problem takes considerably more time than the rest of the assignment. Do NOT leave this until the last minute.

(a) Implement BFS (breadth first search), DFS (depth first search), A*, and IDA* (iterative deepening A*). (10 pts)

Write separate functions called *BFS*, *DFS*, *astar*, and *IDastar*. The different function can (and should) share a lot of code. In your program, nodes should be opaque data structures with operations *get_start*, *get_goal*, and *get_neighbors*. Restrict the number of nodes expanded to 10,000 to avoid very long run times. Consider neighbors in the following order when expanding nodes:

$North \rightarrow East \rightarrow South \rightarrow West$

We suggest using Matlab. You are welcome to use a language other than Matlab, but please verify it with the TAs first. We provide support files for this problem in the following archive:

`http://www.cs.cmu.edu/~ggordon/780/handouts.html`.

To make your job easier, we have provided a Matlab priority queue implementation in the archive. You are free to use this implementation or not, as you see fit. The priority queue has the following functions:

- *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 the smallest numerical priority value). It breaks ties arbitrarily.
- *pq_test*: Check whether an item is already on the priority queue.

In the support file archive, we have also provided three sample 2D maze problems. The mazes are given as an $n \times 4$ adjacency matrix where n is the number of cells in the maze ($n = rows \times cols$). Cells are numbered top to bottom and then left to right, starting with 1 in the top left corner. The *draw_maze* function that we

provide has a flag that will tell it to display indexes in each cell that correspond to rows in the adjacency matrix. The adjacency matrix displays the legal moves possible in a given cell as below where 1's denote legal moves:

Cell	N	E	S	W
1	0	1	1	0
2	1	1	1	0
3	1	0	1	0

We also provide helper function for the mazes:

- *load_maze*: loads a maze into a structure for use with other function.
- *draw_maze*: provides a visualization of the maze.
- *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.

(b) Please run your BFS, DFS, A*, and IDA* implementations on the three sample mazes. Have the starting node be the top left cell: (x, y) coordinates $(1, 1)$, and have the ending cell be the bottom right cell: (x, y) coordinates $(Cols, Rows)$. **Experiment with using TWO different admissible heuristics for A* and IDA***. Which heuristics did you use?

i. Provide a printout of the path (ordered (x, y) coordinates) found by your BFS, DFS, A* (one for each heuristic), and IDA* (one for each heuristic) implementation on maze3.txt. (5 pts)

ii. Record run times, number of nodes expanded, and path length for each search method. Note: you should have run times, nodes expanded, and path length results for BFS, DFS, A* with heuristic 1, A* with heuristic 2, IDA* with heuristic 1, and IDA* with heuristic 2. (5pts)

iii. Compare the search results of the different methods and heuristics. Which methods performed best? Why? Which heuristic performed best? Why? (5 pts)

(c) For this problem, we will expand the state space to 4D to incorporate the dynamics of the agent moving through the world. The agent's state will now consist of (x, y) coordinates as well as (dx, dy) velocities. Instead of controlling its movement directly, the agent now controls its acceleration and deceleration. We have restricted the problem as follows:

- The agent starts in the top left corner $(x, y) = (1, 1)$ with zero velocity $(dx, dy) = (0, 0)$.
- The agent must move to the bottom right corner $(x, y) = (Rows, Cols)$ and slow down to zero velocity.
- The agent can accelerate or decelerate by one in either dx or dy (or remain at the same velocity).
- The agent's velocity in both directions must always remain less than or equal to a maximum velocity provided in the variable $maxV = 2$. Both velocity components must also always remain positive.

- The agent cannot slow down by hitting an obstacle. If it is traveling at a high speed, it must slow down several squares before an obstacle to avoid collision. (You do not want to scratch your car on those pesky obstacles).

For example, if the agent is in state $(x, y, dx, dy) = (1, 1, 2, 0)$, it can choose to speed up dy, slow down dx, or remain at the same speed. It cannot speed up dx because it would break the speed limit, and it cannot slow down dy because it would be moving backwards. If it speeds up dy, it goes to $(x, y, dx, dy) = (3, 2, 2, 1)$. If it slows down dx, it goes to $(x, y, dx, dy) = (2, 1, 1, 0)$. If it remains at the same speed, it goes to $(x, y, dx, dy) = (3, 1, 2, 0)$. For more clarification, see the file *get_neighbors_dynamic* in the archive.

We have provided you with two maps for use in the 4D problem (dynamics_maze1.pgm and dynamics_maze2.pgm). These can be read in using the *read_map_for_dynamics* function. We have also provided you with the functions *test_dynamic_neighbors* and *test_dynamic_path* to help familiarize you with the format of the maps. Note that these maps are in a different format from the maze format above.

For this problem, we have provided you with Matlab code for the functions *get_goal_dynamic*, *get_start_dynamic*, and *get_neighbors_dynamic*. These will replace the corresponding functions which you implemented for the 2D maze. The functions *dynamic_state_from_index* and *dynamic_index_from_state* also provide the same functionality as the corresponding maze functions.

i. Please replace the functions that you coded up for the 2D maze problem above, and run your search algorithms on the 4D problem. **Develop an admissible heuristic for the 4D domain** (you need only use one). What was your heuristic? (5 pts)

ii. Record run times, number of nodes expanded, and path length for each search method on the two maps provided for the 4D problem. Also provide a printout of your path (ordered (x, y, dx, dy) states) for each method on dynamics_maze2.pgm. What has changed from the 2D maze to the 4D maze with dynamics? Which methods are preferable in this new domain? (5 pts)

iii. Compare the performance of A* versus IDA* on the 4D problems. Which had shorter run times? Which expanded fewer nodes? Taking into account these results as well as those of 3.b.iii, in general, when would one prefer to use A* instead of IDA* (and vice versa)? (5 pts).".

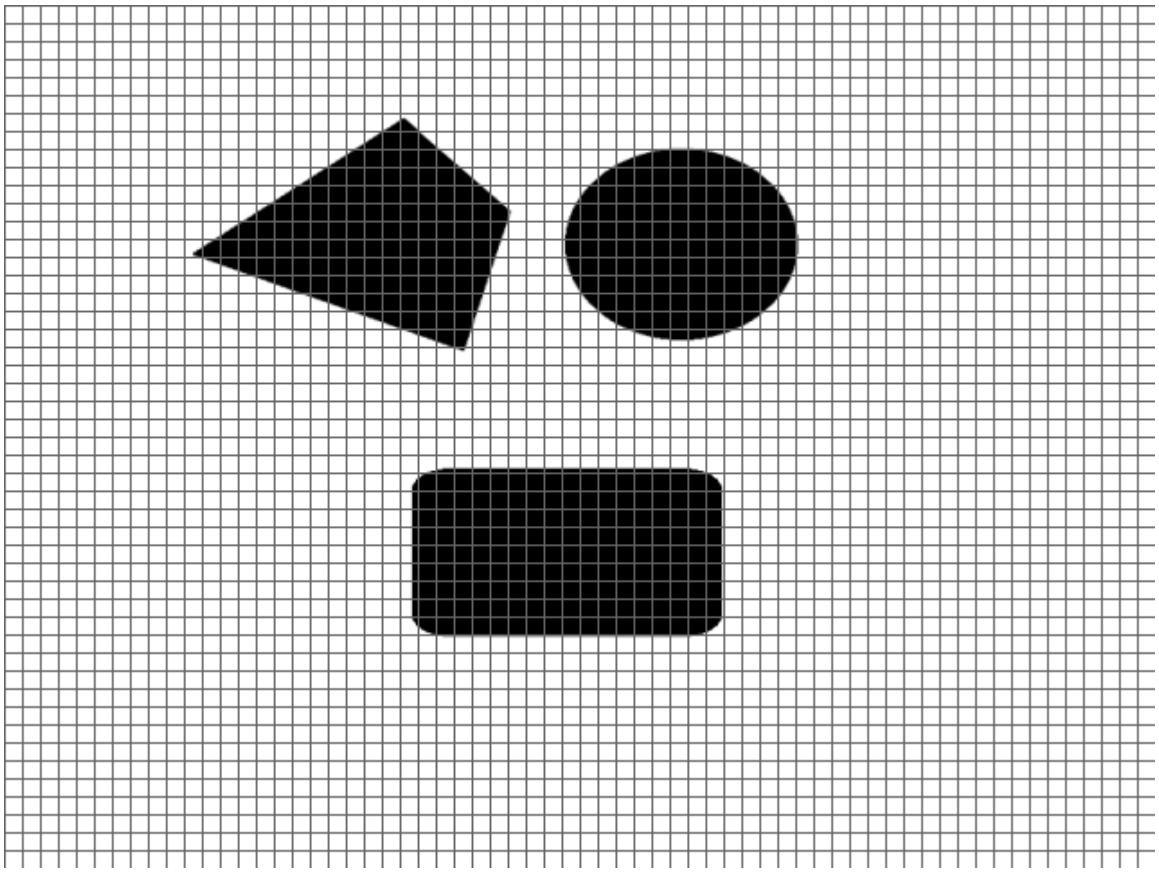

Figure 2: Workspace for problem 2(a).