Graduate Artificial Intelligence 15-780

Homework 3: *Motion Planning & CSPs*

Out on March 5
Due on March 19

Problem 1: Polygonal Planning [John, 10 Pts[1]]

Assume you are in control of a robot with full knowledge of the obstacles in its world. Furthermore, any obstacles are closed polygonal regions. In class, we proved that the shortest path from a starting position to a goal position is a polygonal line with vertices (if they exist) corresponding to vertices on the obstacles.

We now introduce a new distinction: a vertex x whose internal angle formed by its two edges is less than 180 degrees is called *convex*, while a vertex y whose internal angle formed by its two edges is at least 180 degrees is called *concave*. These are the normal definitions of convex and concave (see Figure 1 for an example).

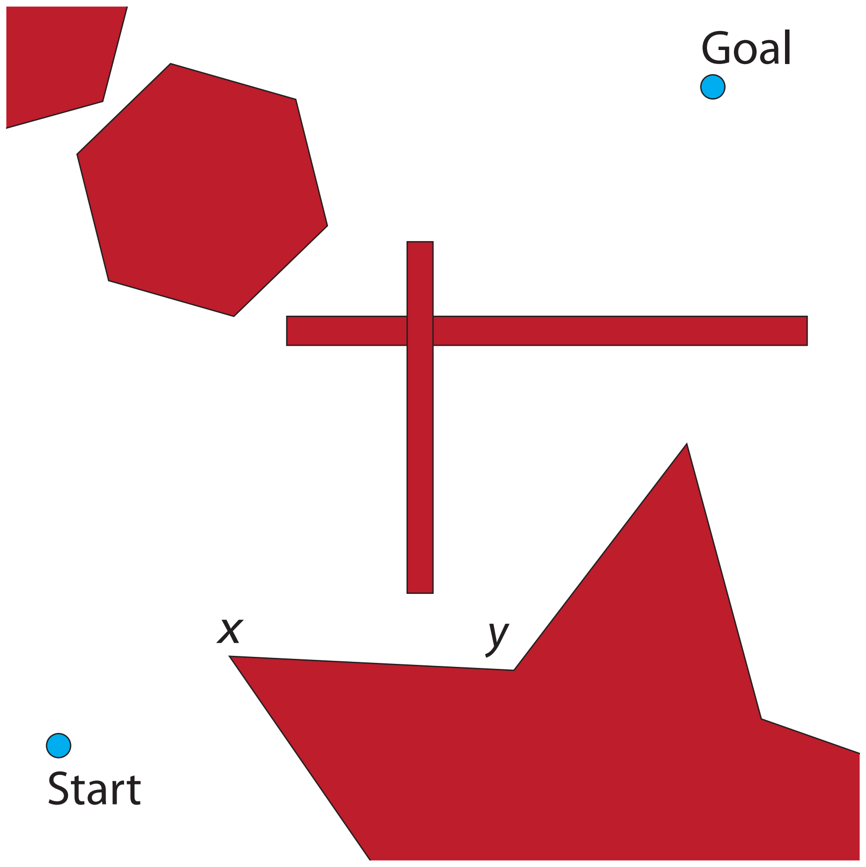

Figure 1: A polygonal world with start and goal state. The vertex x is *convex*, while the vertex y is *concave*.

Using techniques similar to those from class, prove that the shortest path from the start to goal positions is a polygonal line with vertices (if they exist) corresponding to only *convex* vertices on the obstacles.

Problem 2: A Robot in Manhattan [John, 20 Pts[1]]

A robot moves from vertices to vertices in the unbounded regular 2-D grid shown in Figure 2. The initial position of the robot is the vertex $(0,0)$, marked with WALL-E. At each step the robot can move from its current position by a unit increment up, down, left, or right. The goal is for the robot to move to some given grid vertex (x, y).

a) How many distinct positions can the robot reach in $k > 0$ steps that it could not reach in fewer than k steps? Please explain your answer. [2 pts]

   (i) 4^k (ii) $4k$ (iii) $4k^2$

[1] With thanks to Jean-Claude Latombe!

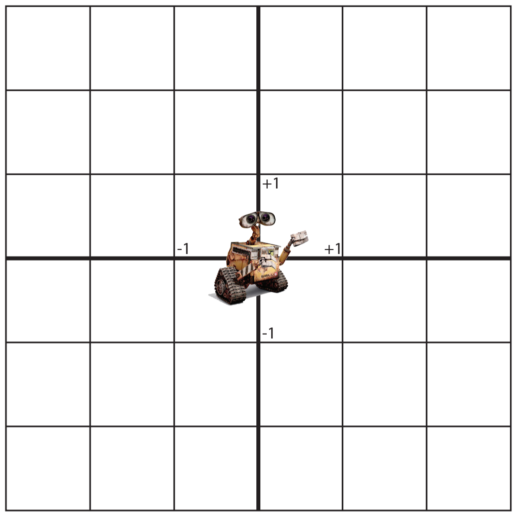

Figure 2: A portion of the grid, with a friendly robot at position $(0,0)$.

b) In the last homework, we discussed *admissible* heuristics. Recall our (more strict) definition of a *consistent* heuristic from lecture. Let the robot be at position (x, y). For a node $n = (u, v)$, is $h(n) = |u - x| + |v - y|$ a consistent heuristic? Why? [6 pts]

c) If we *remove* some edges from the grid, does this heuristic h remain consistent? Why? [6 pts]

d) If we *add* new edges into the grid, does h remain consistent? Why? [6 pts]

Problem 3: New and Improved A* Search [John, 35 Pts]

For the last homework assignment, you coded up a version of the class A* algorithm for use on static 2-D grids. Since then, we've discussed some variants of the A* algorithm, like weighted A* search and ARA*. A brief refresher for these two variants is below; for more information, see the slides from lecture.

Weighted A*

When faced with a difficult search problem, A* often takes too long to find an optimal solution (as many of you know!). In cases such as these, approximate solutions are often more helpful than no solution at all. Weighted A* defines, for a node n, a function $f'(n) = g(n) + w \times h(n)$ for some exogenous weight $w \in \mathbb{R}^{\geq 1}$. This node evaluation function f' is essentially f from normal A*, just with a little more emphasis put on the heuristic.

Anytime Repairing A* (ARA*)

Weighted A* provides a (bounded) suboptimal solution. Anytime Repairing A* (or ARA*) aims to fix this by quickly finding a suboptimal path under a loose optimality bound, and then progressively refining the path by tightening the bound. In lecture, we discussed ARA*'s small additions to the classical A*'s `ComputePath` function, as well as the ARA* `main` loop for refining search. For more information, you will need to read and understand:

- Likhachev, Ferguson, Gordon, Stentz, and Thrun. Anytime Search in Dynamic Graphs. *Artificial Intelligence*, 2008, pp. 1613–1643.

A link to this paper can be found on the course website.

Problems

a) Implement weighted A* using your codebase from the last homework assignment. This should be a trivial extension to your code. Pick a map (*not* the empty map, though) and run an extensive simulation comparing the number of nodes visited, expanded, and the final path lengths returned by weighted A* run on $w \in \{1.0, 1.5, 2.5, 5.0, 10.0\}$. Report this comparison in an easy-to-read table or figure. [10 pts]

b) Implement ARA*. The `ComputePath` function for ARA* is very similar to that used by your current A*. Using the map you chose above (again, *not* the empty map), run a similar experiment, recording the nodes visited, expanded, and the final path lengths at each iteration of the weight w, with the initial $w = 10$, in ARA*. How do the two algorithms compare qualitatively and quantitatively? [25 pts]

Problem 4: A Scheduling Nightmare [John, 35 Pts]

As an overworked student, you must complete all of your homework assignments in each of your classes before they are due at the end of the semester (at time $t = T$). Each homework assignment has a specific duration $d \in \mathbb{N}$ representing how long it will take to complete. You must also complete them in order (so the first homework assignment in a class must be completed before the second in that same class); however, homework assignments in one class can be completed in any order relative to assignments in another class.

Since you are a student in computer science, many of your homework assignments require the use of specific machines for experimentation. You have a set of k computers; some homework assignments require one or more specific computers, and occupy them entirely for the entire duration of the homework assignment (i.e., if both homework A and homework B require computer 1, then you cannot work on A and B at the same time).

An Example Semester

Figure 3 gives an example semester-long homework schedule for a student. The student has four classes, and each class has either three or four homework assignments. These homework assignments have (i) a duration in time periods, (ii) a relative ordering to other homework assignments in the same class, and (iii) an optional resource requirement. We assume the semester is 15 time periods long.

For example, the homework $H_{0,1}$ has duration 4 and requires resource 0. Similarly, homework $H_{1,0}$ has duration 1 and also requires resource 0. As discussed above, the shared resource constraint would prevent the student from overlapping these two homeworks in her schedule.

Problems

Please use the example from Figure 3 in these problems.

a) What are the variables and their domains? [2 pts]

b) Describe how you would write this problem as a CSP with arity $r > 2$, e.g., with at least some constraints having at most r variables. [4 pts]

c) Qualitatively, how would this change if you were limited to arity $r = 2$? Recall from class that any CSP can be written this way. [4 pts]

d) Solve your scheduling CSP—either the $r > 2$ or $r = 2$ version—by hand or with code using (i) a deterministic ordering of variables and values (e.g., alphabetical order) and (ii) the most constrained variable and least constraining value heuristics discussed in class. How much of an effect did these heuristics have on the size of the search tree? Quantify your answer. [15 pts]

e) The problem above is solved relatively easily with simple heuristics and backtracking search. This is not always the case. Come up with a short "worst-case" instantiation of the general scheduling problem. What are some properties of these search problems that will thwart the most constrained variable heuristic? The least constraining value heuristic? [10 pts]

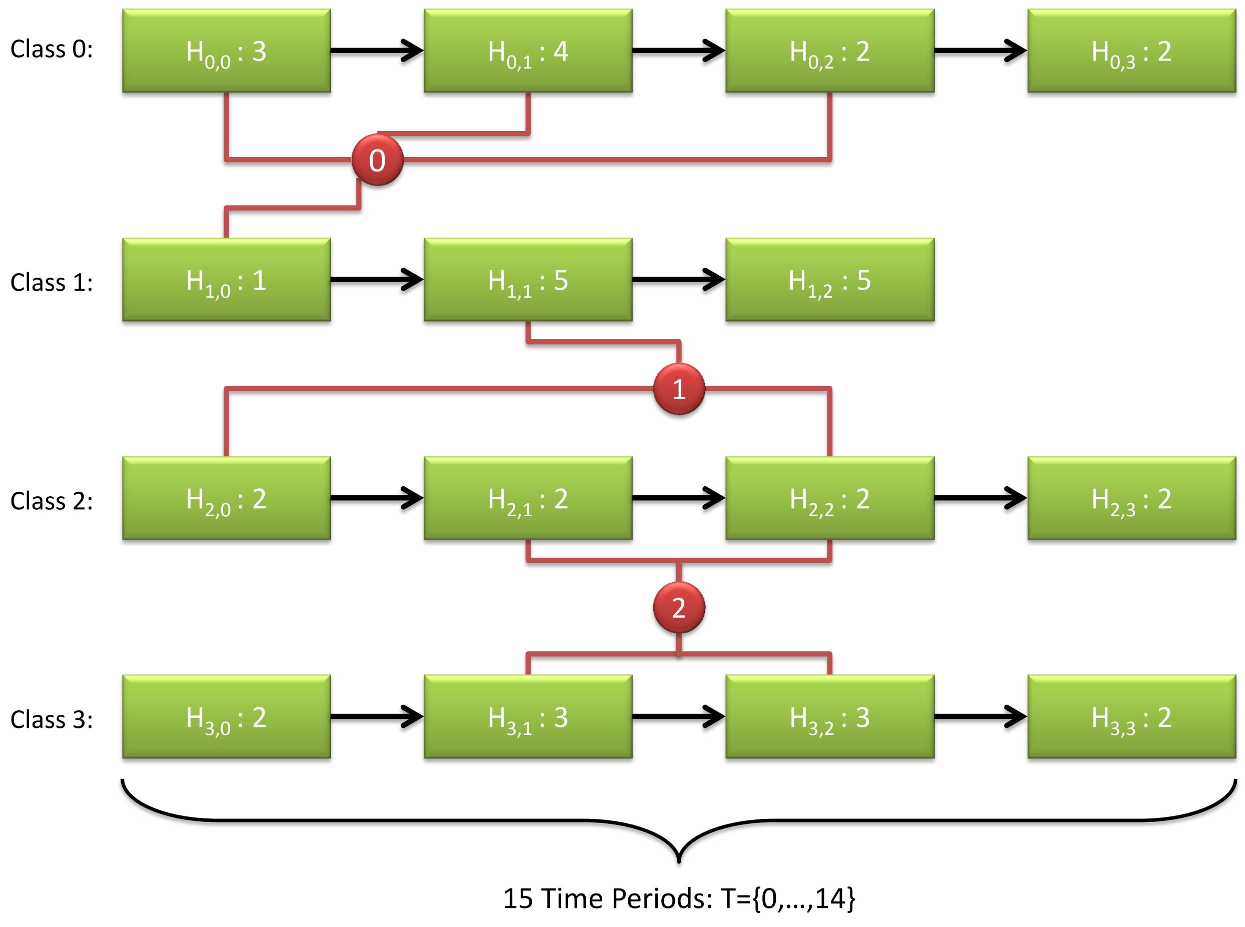

Figure 3: A semester's worth of homeworks for four classes. Each homework has a relative ordering (e.g., $H_{0,0}$ must be completed before $H_{0,1}$), an integral duration, and (possibly) a resource constraint. The semester is assumed to be 15 time periods in length.