

CS 221

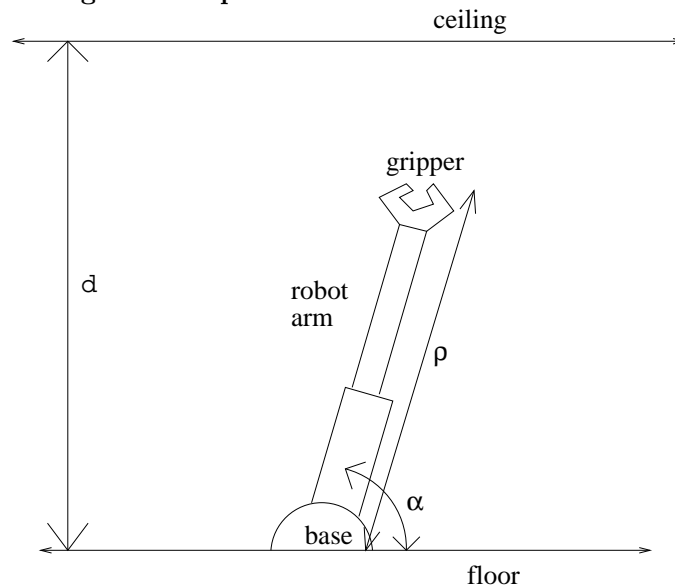
Problem Set #1: Search, Motion Planning, CSPs

Due by 9:30am on Tuesday, October 13. Please see the course information page on the class website for late homework submission instructions. SCPD students can also fax their solutions to (650) 725-1449. We will not accept solutions by email or courier.

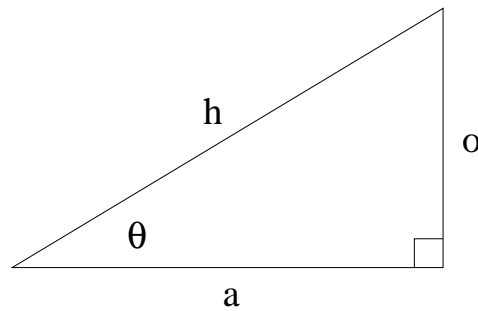
Written part (70 points)

NOTE: These questions require thought, but do not require long answers. Please try to be as concise as possible.

1. [10 points] Configuration Spaces



Consider the robot arm pictured above with two degrees of freedom, operating in a two dimensional workspace. The robot arm has a revolute joint and a prismatic joint. The revolute joint has a range of $0 \leq \alpha \leq \pi$, where α is the angle of the arm relative to the floor. The prismatic joint has a range of $\rho_{min} \leq \rho \leq \rho_{max}$, where ρ is the length of the arm from the base to the gripper. The ceiling is a distance d from the floor, with $\rho_{min} < d < \rho_{max}$. The width of the arm and gripper may be considered negligible.



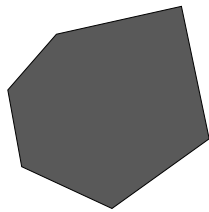
$$\sin \theta = o/h \quad \cos \theta = a/h$$

Draw the configuration space of the robot arm, *using α and ρ as the coordinates of the configuration space* (i.e., the horizontal and vertical axes in your figure should be labeled α and ρ).

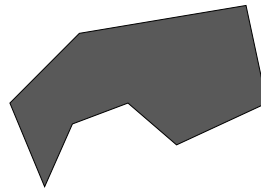
Please specify the coordinates of the important points of the obstacles in configuration space (e.g. leftmost point, rightmost point, etc.). Also, while a freehand drawing is acceptable, please make sure that the shape of the obstacle is clear from your drawing.

2. [14 points] Optimization Search / Configuration Spaces

We consider the problem of moving a robot in various two dimensional (2D) configuration spaces with obstacles. We start with a simple configuration space, with only convex obstacles, and then make it more complicated by allowing non-convex obstacles as well.



convex obstacle



nonconvex obstacle

A discrete search space for this planning problem can be defined using the *visibility graph method*. In this method, we place landmarks in configuration space at the initial position of the robot, the goal position of the robot, and the *vertices* of the polygonal obstacles. Further, the search operators only allow the robot to walk in a straight line between two of these points: a state s in the search space is connected to any other state s' which can be reached from s by walking along a straight line either completely in free space or along the boundary of an obstacle (i.e., s is connected to s' if s' is “visible” from s).

Assume that our robot navigates this space using a simple greedy hill-climbing search with a straight-line distance heuristic function. (Note that the robot climbs “down” in that the heuristic function decreases as the robot approaches the goal. Nevertheless, we’ll continue to use the “climbing” metaphor, as we did in class.)

We assume that the robot is allowed to move in any direction, but is not allowed rotation.

- (a) [7 points] For this part, assume that the robot is point-sized, so that the configuration space is a 2D space which is identical to the workspace. A local maximum is a

state that is closer to the goal than any of its successors. If the workspace contains only convex polygonal obstacles, is it possible for the robot to get stuck on a local maximum? Either explain why not, or present an example.

- (b) [7 points] Again assume that the robot is point-sized. If we allow arbitrary polygonal obstacles, can there be any plateaus in this space? Either explain why not, or present an example.

For this problem, define a plateau to be a contiguous region of 4 points or more where the points all have the same heuristic value (so that it is locally impossible to determine the direction to the goal). There can be other neighboring points where you “fall off the edge” of the plateau, so that the heuristic value is strictly worse. But the plateau itself has to consist of a contiguous set of points with the same value.

3. [18 points] **Search Space Formulation** Consider the problem of a student planning an entire course of studies at Stanford in advance. The desired outcome is an assignment of classes to quarters (1st quarter, 2nd quarter, etc.) which satisfies the requirements of the major. To simplify the problem, assume the following:

- There is some set of classes C such that the student must take each of the classes in C exactly once (i.e., there are no alternatives of the form: you must take either CS106A and CS106B or CS106X).
- For each class c_i , there is a (possibly empty) set $P_i \subseteq C$ of prerequisite classes that the student must take strictly before s/he takes c_i .
- The student must take at least one and at most four classes each quarter.

- (a) [8 points] Provide a detailed representation of your search space. Your description must be precise, i.e. you should provide an exact specification of the components of the state description (state space, initial state, operators, goal test), of the constraints under which each operator can be applied, and of the effects of each operator on the state components. You may use either English or pseudocode.

Make sure that your formulation of the search space has no problem with repeated states, i.e., that no node can be reached from the initial state by two different paths.

- (b) [3 points] Somewhat shortsightedly, the student’s only aim is to complete the degree in the shortest possible time, thereby hopefully minimizing the total amount of money spent on tuition. Tuition is charged per quarter, not per unit, and all quarters cost the same amount. Specify a cost function for the operators in your space so that the cost of a solution will correspond exactly to the student’s preferences.
- (c) [7 points] Construct a reasonable and admissible heuristic for this search space, given your costs in 3b. (For example, a zero heuristic function is admissible but not reasonable; for different reasons, a heuristic function that computes the optimal schedule is not reasonable either.) Provide an exact description of your heuristic function, and explain why it’s admissible, why it’s a reasonable heuristic, and how you would compute it efficiently given a state in your search space. (Hint 1: As discussed in lecture, try relaxing some of the constraints on your operators. Hint 2: There are multiple, equally good answers to this problem.)

4. [16 points] **Approximately Admissible A***

Assume we are given a heuristic function which is only approximately admissible. I.e., there exists some $\epsilon > 0$ such that for every node n in the tree, we have $h(n) \leq h^*(n) + \epsilon$.

(Note that in this case it is possible for $h(n)$ for a *goal* node n to be bigger than zero.) Let n^* be the optimal goal node, with cost $g(n^*)$.

Assume that we run standard A* with our ϵ -admissible heuristic h . Let n_g be the goal node that A* returns. Prove that the cost $g(n_g)$ is at most $g(n^*) + \epsilon$. You may assume that the function $f = g + h$ corresponding to the heuristic function h is monotonic (and therefore that A* expands nodes ordered in increasing values of f). Hint: You may use any result which we have proved during the course. This should be about a five-line proof.

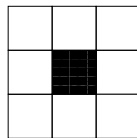
5. [12 points] **Constraint Satisfaction with Non-Binary Constraints**

In CSPs, a non-binary constraint is a constraint that involves more than 2 variables. For example, consider the variables X, Y, Z where the domain of X and Y is $\{Red, Green\}$ and the domain of Z is $\{Red, Green, Blue\}$. An example of a non-binary constraint is that exactly two of these variables have the color *Green*.

Sometimes we want to convert a non-binary CSP to a binary CSP. To do that we add a new extra variable for every constraint.

In detail, to replace a non-binary constraint C (over variables X_1, \dots, X_k) with only binary constraints, we create a new variable W . The variable W can take on as many values as there are legal assignments to X_1, \dots, X_k that satisfy C . We then remove the original constraint, and replace it with a set of binary constraints between the new variable W and the original variables X_i participating in the original constraint. For simplicity, you may assume that the original CSP does not contain any unary constraints.

- [3 points] Complete the definition for a transformation of this type when it is executed for a constraint C over a set of variables X_1, \dots, X_k . You may assume that C is represented as a set of legal tuples of values for X_1, \dots, X_k . Specifically, how would you define W , and what are the binary constraints associated with W ?
- [3 points] Apply the algorithm to the example given above (where exactly two variables out of X, Y, Z have the color *Green*). Define exactly the domain of the new extra variable by writing down all the possible values in the domain. Show the new constraints created by writing down all the constraints that involve Z and the new extra variable.
- [6 points] Consider the problem of creating a crossword puzzle in which all the words must appear in some given dictionary D . We use the notation $D = \{w^1, w^2, \dots, w^n\}$, where w^i is the i -th word in the dictionary. Furthermore, we use the notation $w^i = w_1^i, w_2^i, \dots, w_{l_i}^i$ where w_j^i is the j -th letter of the i -th word and l_i is the length of w^i . As an example, consider the following instance of the problem:



Assuming that $CAP, CAT, PEN, TIN \in D$ (which can, of course, also contain other

words), a possible solution could be:

C	A	P
A	■	E
T	I	N

- i. **[3 points]** Formalize this instance of the problem as a CSP problem, such that each variable corresponds to one empty square in the crossword puzzle (you will need 8 variables). What are the constraints in this formalization?
- ii. **[3 points]** Note that the constraints are non-binary. If we reformulate the problem as a binary CSP using the method from the previous parts, the extra variables and their domains have very intuitive meanings. What are they? (This answer should not take more than one sentence!)