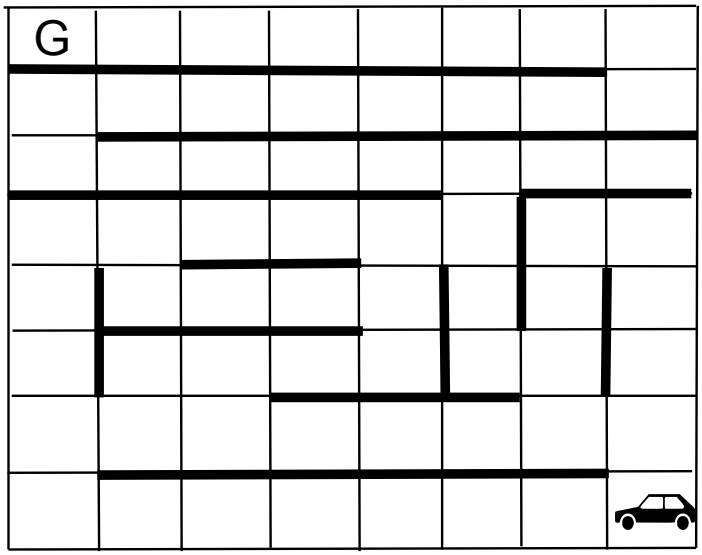
Artificial Intelligence Homework 1
Due at the start of class, in hard copy, on 9/20/2012

1. Imagine a car-like agent wishes to navigate a maze such as this one:



The agent is directional and at all times faces some direction $d \in \{N, S, E, W\}$. With a single action, the agent can either move forward at an adjustable velocity v, or turn.

The turning actions are LEFT and RIGHT, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero).

The moving actions are MOVE, MOVE-FASTER and MOVE-SLOWER. MOVE moves the car in its current direction forward by the number of squares equal to its velocity. MOVE-FASTER increments the velocity by 1 and then moves forward the number of squares equal to its new velocity. MOVE-SLOWER decrements the velocity by 1 and then moves forward the number of squares equal to its new velocity.

Any action that would result in crashing into a wall is illegal (meaning it is not available as an action from the state in question). Any action that would reduce v below 0 or above some maximum speed V_{max} is also illegal.

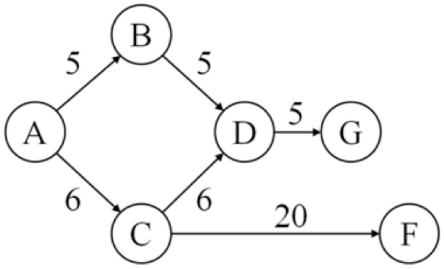
The agent's goal is to find a driving strategy that parks it (stationary, with $v = 0$, direction facing irrelevant) on the goal square G in as few actions (time steps) as possible. Every action (turning 90 degrees, or driving forward at the current velocity) takes 1 unit of time.

Note that this is problem formulation is different from the regular navigation problem!

All of these questions pertain to a generic maze of size *m* by *n*, not specifically the maze above, which is only shown to illustrate how the problem is set up.

(a) If the grid is *m* by *n*, what is the (maximum) size of the state space? Justify your answer. You should assume that all configurations are reachable from the start state.
(b) What is the maximum branching factor of this problem? You may assume that illegal actions are simply not returned by the successor function (the function that, given a state, returns the set of possible actions from that state). Briefly justify your answer.
(c) Is the Manhattan distance from the agent's location to the goal location admissible? Why or why not?
(d) Describe and justify a non-trivial admissible heuristic for this problem that is not the Manhattan distance to the goal.
(e) If we used an inadmissible heuristic to solve this problem, could it change the completeness of the search? Why or why not?
(f) If we used an inadmissible heuristic to solve this problem, could it change the optimality of the search? Why or why not?

2. Consider the following search problem, given as a graph:



Four possible heuristics:

	A	B	C	D	F	G
h1	0	0	0	0	0	0
h2	11	7	7	3	0	0
h3	13	9	7	1	0	0
h4	15	10	11	5	0	0

(a) Which, if any, of the heuristics are admissible?
(b) Which, if any, of the heuristics are consistent?
(c) Run A* (by hand) using h3, showing the order that nodes are considered, along with keeping track of the frontier and explored lists. This is probably easiest by constructing a table with each row corresponding to a step of the algorithm, keeping columns for which node is being expanded, which nodes are on the frontier, and which states are on the explored list. For the nodes on the frontier and the one currently being expanded by A*, make sure to list out the entire path so it's clear that, for instance, you're examining the path ABD rather than ACD. What is the best path returned by A*?

(d) Run greedy best first search by hand using h3, following the same procedure as above. What is the best path returned by greedy best first search?

3. We can use the local search framework for CSPs by generating a random assignment of values to variables and counting the number of violated constraints (trying to get that number of violations to go down to zero). Actions correspond to changing the value of a variable to a different value (hopefully reducing the number of constraint violations). The n-queens problem can be solved this way by generating an initial random placement of queens on the board (1 per column) and then adjusting each queen within its column to minimize the number of conflicts that that particular queen has with the other queens on the board (see section 6.4 for a diagram and a longer explanation). Using this formulation of the n-queens problem, local search solves the problem very quickly.

   Explain why local search for n-queens using hill climbing (even allowing sideways moves) can get stuck in a local maximum but local search using the CSP formulation won't get stuck (and will eventually reach the global maximum).

4. Sam is an ambitious undergraduate CS student entering spring of junior year. Sam needs to take four classes in the remaining three semesters: Advanced Algorithms (A), Beginning Operating Systems (B), Computer Organization (C), and Databases (D).

   There are some restrictions on these courses. Specifically:
   • Advanced Algorithms and Databases are co-requisites (meaning they must be taken during the same semester);
   • Advanced Algorithms and Basic Operating Systems are always offered during the same time of day, and therefore cannot be taken simultaneously;
   • Computer Organization is a pre-requisite for both Databases and Basic Operating Systems, and therefore must be taken in a preceding semester.

   We will now formulate Sam's scheduling problem as a constraint satisfaction problem in which each variable corresponds to a class Sam must take to graduate, and the values correspond to a numbering of the remaining semesters: 1 for the spring of junior year, 2 the fall of senior year, and 3 the final spring.

   (a) State the domains of the variables and the constraints in mathematical notation using the following symbols: $=, \neq, <, >, \leq, \geq$ (Assume that each class is offered every semester.)

   We will now solve the constraint satisfaction problem formulated in (a) using backtracking depth-first search and arc consistency. Run arc consistency (AC-3) as a preprocessor *and* after each variable assignment. When there is ambiguity as to which variable to assign next, choose the variable with the letter earliest in

the alphabet. Furthermore, when the next value to assign to a variable is ambiguous, break ties by assigning the earliest possible semester first.

(b) What are the remaining domains after enforcing arc consistency on the initial CSP with no assignments? Show the steps of AC-3 as clearly as possible.

(c) What are the remaining domains after assigning A = 2 and re-enforcing arc consistency?

(d) State a solution to the CSP.

(e) True / False: With arc consistency, we will never have to backtrack while solving a CSP. Justify your answer.