15-780: Graduate AI

Homework Assignment #1 Solutions

Out: January 22, 2015
Due: February 4, 2015 5 PM

Collaboration Policy: You may discuss the problems with others, but you must write all code and your writeup independently.

Turning In: Please email your assignment by the due date to `shayand@cs.cmu.edu` and `vdperera@cs.cmu.edu`. If your solutions are handwritten, then please take photos or scan them and make sure they are legible and clear. Please submit your code in separate files so we can easily run them.

Note: We use the term nontrivial heuristic in this homework to mean a heuristic function that does not assign a constant value to every node.

1 Streets of Pittsburgh [40pts]

In this question, you will try a variety of search algorithms you learned about in class to be able to find the shortest path between any two intersections in Pittsburgh. You will be given real data exported from OpenStreetMap. OpenStreetMap labels each intersection, called a **node**, with a unique ID. (You can see where each nodeID corresponds to by going to: `http://www.openstreetmap.org/node/nodeID`.) A street is therefore an **edge** between two nodes. All the streets in a large part of Pittsburgh are given to you in `edges.txt`. Each line specifies a single weighted undirected edge in the following format:

`node1ID node2ID distance_in_km`

1. **Programming** Write a program that can find a path between any two nodes using breadth-first search, uniform-cost search, iterative deepening, and A*. In particular write four functions as follows:
   `bfs(start_nodeID, goal_nodeID)`, `ucs(start_nodeID, goal_nodeID)`,
   `iterdeep(start_nodeID, goal_nodeID)`, `astar(start_nodeID, goal_nodeID)`.
   Each of these functions should read in the file `edges.txt` from the same directory as your code. When choosing which node to visit, please break ties randomly. Your code should output a string that tells us (a) how many nodes were visited (i.e. were popped

from the queue), (b) how many nodes were on the path that was chosen, and (c) the distance of the path chosen (in kilometers). Please format the string to be as follows:

```
Num nodes visited:  ?
Num nodes on path:  ?
Distance (km):  ?
```

For A*, your heuristic function will be the Euclidean distance to the goal node. The `astar` function should read in a file `heuristic.txt` from the same directory as your code. Each line in this file gives you the distance from a particular node to a some goal node in the following format:

```
nodeID distance_to_goal_in_km
```

The `heuristic.txt` file we provide you with the homework provides distances to the Pittsburgh City Hall (node 105012740). You can test your code using that goal node, however we may use another goal node (and another heuristic file) when testing your code.

You may write your code in any commonly used language, but please do not use any graph libraries. Make sure to submit all of your code so we can run it!

2. **Traffic Time!** Now imagine instead of finding the path with the shortest distance, you wanted to find the path that takes the shortest time during current traffic, which you can assume is information you have access to. For this problem you may assume drivers never drive faster than the speed limit. ☺ Give a nontrivial admissible heuristic for this new problem.

   One possible admissible heuristic is the Euclidean distance divided by the maximum speed limit on any street.

3. **Real-World Routing** You shouldn't use the solutions you get from your code for driving just yet! What is one thing that is missing in the way this problem is currently formulated that makes it unusable in the real-world? Suggest an easy way to solve the problem.

   One possible answer is that we do not consider one-way streets, so a lot of the edges in the graph should actually not be there! An easy solution to this is to make the graph directed.

2 Heuristicars [15pts]

Imagine an $n \times n$ grid where there is a car in each of cells $(1, 1)$ to $(n, 1)$—the bottom row. Every time step, each car can move up, down, left, right, or stay put. If a car does not move, at most one other car can hop over it. At most one car can be in a cell at any given time. Cars move simultaneously (so a car can move into the place of another car if the other car moves out), and two cars are not allowed to move into the same cell. Moving a car in any direction (including hopping over another car) incurs a cost of 1 whereas staying put has no cost. The total cost at any time step is the sum of the cost for each car. The goal is to move all the cars to the top row but in reverse order so that the car starting in cell $(i, 1)$ ends up in cell $(n - i + 1, n)$.

1. What is the size of the state space in big-O? $O(n^{2n})$
   The number of ways to place n distinct objects in n^2 cells is
   $$n^2(n^2 - 1) \cdots (n^2 - n + 1) \in O(n^{2n})$$

2. What is the worst-case branching factor? 5^n
   Each car has at most five possible moves from any position. Note that if each car has a 3x3 grid of empty space around it, then all five moves are possible for each car and they will result in unique configurations (since the cars won't interact). When n is large enough you have enough 3x3 grids to support all possible cars. Consider when $n = 12$. There are 16 3x3 grids, so we can support all 12 cars and even the next two as we increase the size of the grid. (We can actually support the next four, but that's not necessary.) Each time we increase n by 3, we can clearly support at least two cars beyond the size of the grid (i.e. we can clearly support at least 14 cars with a 12x12 grid, 17 cars with a 15x15 grid and so on), so the worst-case branching factor is attainable for at least all $n \geq 12$. (Note: You don't have to actually argue that you can place the cars in this way for full credit.)

3. Suppose there was only one car starting at $(i, 1)$ and the problem was to only move that car to its goal location. Give a nontrivial admissible heuristic $h_i(x_i, y_i)$ for moving car i from cell (x_i, y_i) to its goal at $(n - i + 1, n)$, and explain why it is admissible. Would it be admissible when the other $n - i$ cars are also at various positions on the grid?

   We can use manhattan distance, i.e. $h_i = |(n-i+1) - x_i| + |n - y_i|$. This in fact gives the exact cost of moving the car to the goal, so it is also admissible. If there were other cars on the grid, this would no longer necessarily be an admissible heuristic because car i could potentially jump over other cars, allowing it to get to the goal in less steps than just using Manhattan distance.

4. Which of the following are admissible heuristics for the original problem and why? (Note: h_i is the heuristic you came up with in the previous part.)
   (a) $\sum_i h_i$ (b) $\min(h_1, \ldots, h_n)$
   (c) $\max(h_1, \ldots, h_n)$ (d) $n \min(h_1, \ldots, h_n)$
   (e) $n \max(h_1, \ldots, h_n)$

   Only (b) is admissible.
   Suppose car i has the minimum heuristic $h_i = \min(h_1, \ldots, h_n)$. If car i is at its destination, the heuristic value is 0, which is also the exact cost, and hence admissible. Otherwise, notice that every time car i hops over another car, the car it hops over

cannot be at its destination, because if it was at its destination, $\min(h_1, \ldots, h_n) = 0$, which is a contradiction. Thus the car that is hopped over must have at least one step remaining to reach its destination. Therefore the total cost of moving all cars to their destinations is at least h_i minus the number of hops car i makes plus 1 for each hop, which is h_i.

(c) is not admissible. Consider the following state:

	2	
1		
3		

Here we see that if car 3 hops twice, the total cost of moving all the cars to their destinations is 3, whereas the heuristic $\max(h_1, \ldots, h_n) = 4$.

(d) is not admissible. Consider the following state:

2		
1		3

Here we see that if car 1 hops once, the total cost of moving all the cars to their destinations is 5, whereas the heuristic $n \min(h_1, \ldots, h_n) = 6$.

(a) and (e) are not admissible since $\sum_i h_i \geq \max(h_1, \ldots, h_n)$ and $n \max(h_1, \ldots, h_n) \geq \max(h_1, \ldots, h_n)$ respectively.

3 Convex Conundrums [15pts]

This question will ask you to prove or disprove whether certain sets are convex. You should use the definition of convex sets. Please give a formal (but not necessarily lengthy proof) for each. To show a set is not convex, you need to show that there exist two points x and y that are within the set but for some $0 \leq \lambda \leq 1$, $\lambda x + (1-\lambda)y$ is not in the set.

1. The norm cone, i.e. $\mathcal{C} = \{(x,t) | \|x\| \leq t\}$

   The set is convex.

   If $(x,t), (y,u) \in \mathcal{C}$, then $\|x\| \leq t$ and $\|y\| \leq u$. By the definition of the norm we also know that $\|\lambda x\| = \lambda \|x\| \leq \lambda t$ and $\|(1-\lambda)y\| = (1-\lambda)\|y\| \leq (1-\lambda)u$ for all $\lambda \in \mathbb{R}$. Finally, using the definition of the norm again, we have that $\|\lambda x + (1-\lambda)y\| \leq \|\lambda x\| + \|(1-\lambda)y\| \leq \lambda t + (1-\lambda)u$, meaning $(\lambda x + (1-\lambda)y, \lambda t + (1-\lambda)u) \in \mathcal{C}$ for all $\lambda \in [0,1]$, thus proving the convexity of $\mathcal{C}$.

2. $\mathcal{C} = \{x + y | x, y \in \mathcal{D}\}$ for some convex set $\mathcal{D}$

The set is convex.

Consider some $w, z \in \mathcal{C}$. By definition of $\mathcal{C}$, we have that $z = z_1 + z_2$ and $w = w_1 + w_2$, for $z_1, z_2, w_1, w_2 \in \mathcal{D}$. Thus for any $\lambda \in [0, 1]$, we have that

$$\lambda w + (1-\lambda)z = \lambda(w_1 + w_2) + (1-\lambda)(z_1 + z_2) = \lambda w_1 + (1-\lambda)z_1 + \lambda w_2 + (1-\lambda)z_2$$

Since $\mathcal{D}$ is convex, we know that $\lambda w_1 + (1-\lambda)z_1 \in \mathcal{D}$ and $\lambda w_2 + (1-\lambda)z_2 \in \mathcal{D}$. Therefore we know that

$$\lambda w_1 + (1-\lambda)z_1 + \lambda w_2 + (1-\lambda)z_2 \in \mathcal{C}$$

thus proving the convexity of $\mathcal{C}$.

3. The intersection of m convex sets, i.e. $\mathcal{C} = \bigcap_i \mathcal{C}_i$, where $\mathcal{C}_i$ is convex for $i = 1 \dots m$

   The set is convex.

   Consider some $x, y \in \mathcal{C}$. By definition of $\mathcal{C}$, we know that $x, y \in \mathcal{C}_i$ for all $i = 1 \dots m$. By the convexity of each $\mathcal{C}_i$, we know that for all $\lambda \in [0, 1]$ and for all $i = 1 \dots m$, $\lambda x + (1-\lambda)y \in \mathcal{C}_i$. Therefore, by the definition of $\mathcal{C}$, we have that $\lambda x + (1-\lambda)y \in \mathcal{C}$, thus proving the convexity of $\mathcal{C}$.

4. The union of m convex sets, i.e. $\mathcal{C} = \bigcup_i \mathcal{C}_i$, where $\mathcal{C}_i$ is convex for $i = 1 \dots m$

   The set is **not** convex.

   Suppose $\mathcal{C}_1 = \{x\}$ and $\mathcal{C}_2 = \{y\}$ and $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ for some $x, y \in \mathbb{R}^n$. Clearly since $\mathcal{C} = \{x, y\}$, we know that $0.5x + 0.5y \notin \mathcal{C}$, thus proving that $\mathcal{C}$ is not convex.

4 Knowledge Representation [15 pts]

Propositional Logic In the not too distant future, a team of three CoBots is devising to split up delivering mail to NSH and GHC. Here are some possible rules for their deliveries:

(a) If CoBot A delivers mail to NSH and does not deliver mail to GHC then either CoBot B delivers mail to GHC or CoBot C delivers mail to GHC.

(b) Either CoBot A does not delivers mail to NSH and CoBot B does deliver mail to NSH or CoBot A delivers mail to NSH and CoBot B does not deliver mail to GHC.

(c) If CoBot B delivers mail to NSH, it is necessary that CoBot A delivers mail to GHC and either CoBot C delivers mail to NSH or CoBot A delivers mail to NSH.

(d) If CoBot B delivers mail to NSH and Cobot A does not deliver mail to GHC then CoBot C delivers mail to either NSH or GHC and either CoBot B does not deliver mail to GHC or CoBot A delivers mail to NSH.

1. Encode statements (a)-(d) in propositional logic using the following literals:

   A: Cobot A delivers mail to NSH
   B: Cobot A delivers mail to GHC
   C: Cobot B delivers mail to NSH
   D: Cobot B delivers mail to GHC
   E: Cobot C delivers mail to NSH
   F: Cobot C delivers mail to GHC

   (a) $A \land \neg C \land (F \lor \neg D)$
   (b) $C \implies E \land F \land (A \lor B)$
   (c) $C \implies B \land E$
   (d) $A \implies (E \lor F) \land (\neg D \lor A)$

2. Transform the statements you just wrote to Conjunctive Normal Form (CNF).

   (a) Already in CNF
   (b) $(\neg C \lor E) \land (\neg C \lor F) \land (\neg C \lor A \lor B)$
   (c) $(\neg C \lor B) \land (\neg C \lor E)$
   (d) $\neg A \lor E \lor F$

3. Using propositional tableaux prove that $\{A \leftrightarrow B, A \lor B\}$ entails $A \land B$.

4. (a) Consider the following statements:

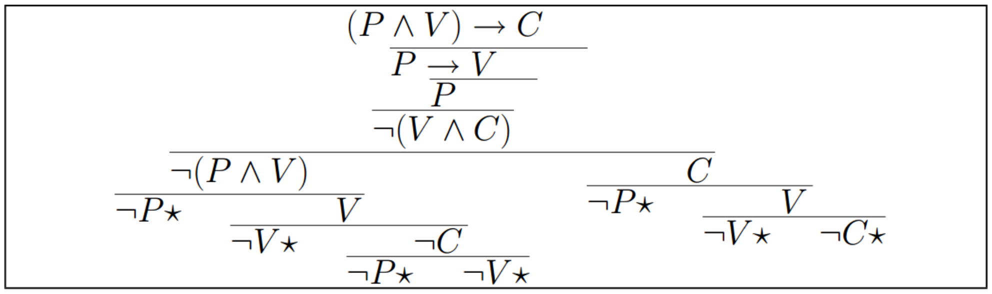
   - If I leave and go on vacation then I am happy.
   - If I leave then I go on vacation.
   - I leave.

   Can we conclude that: I go on vacation and I am happy?

   Yes, let's see how. First we need to encode each statement in propositional logic:

   - If I leave and go on vacation then I am happy.
     (P∧V)→C
   - If I leave then I go on vacation.
     P→C
   - I leave.
     P
   - I go on vacation and I am happy
     V∧C

   Now we can build the tableau but we need to negate the last statement.



(b) Consider the following statements

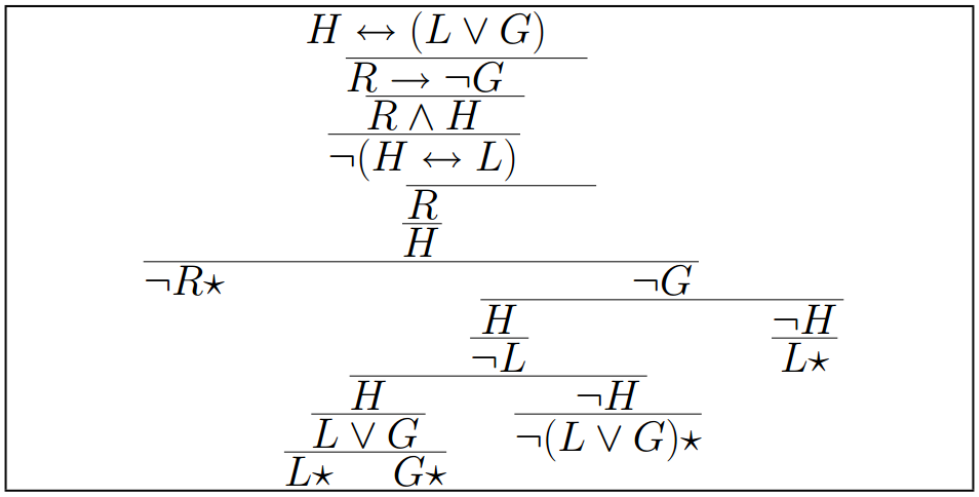
- I am happy if and only if I won the lottery or my friend is with me.
- If it is raining my friend is not with me.
- It is raining and I am happy.

Can we conclude that: I am happy if and only if I won the lottery?

Yes, similarly to what we have done before we rewrite each statement in propositional logic

- I am happy if and only if I won the lottery or my friend is with me.
  H $\leftrightarrow$(L$\wedge$G)
- If it is raining my friend is not with me.
  R $\rightarrow \neg$G
- It is raining and I am happy.
  R$wedge$H
- I am happy if and only if I won the lottery
  H$\leftrightarrow$L

The corresponding tableau is:



5 Constraint satisfaction problem (CSP) [15pts]

A common problem at universities is to schedule rooms for exams. The basic structure of this problem is to divide the exam period into predefined time slots and then assign courses to the available rooms at particular time slots. The assignments must be consistent with a set of constraints. For this question, you will describe how you would design various elements of a CSP program that would solve a restricted version of this problem.

For this version of the problem, a course is defined as having a name, instructor(s) and students. A room has a name and a capacity. Time slots are positive integers, and we will assume that there are three time slots that are represented as 0, 1, and 2. The constraints that must be observed are the following:

- All courses must be assigned to a room and time slot.
- No assigned room can have less capacity than the number of students in the assigned course.
- No room can be double-booked with more than one course per time slot.
- No instructor or student can be in two different rooms during the same time slot.

Using the backtracking-search algorithm presented in class as a framework for your CSP solver, provide concise answers to the following:

There are multiple acceptable answers for this problem, but they should be similar to the example below.

1. Describe the basic data structures you would use to represent the domain objects and the constraints.

   The basic data structures are the following:

   - *person* - represents either an instructor or a student. It should have an attribute for the courses in which the person participates.
   - *course* - represents a course and should have atributes for the instructors, students, and value assignment. The courses are the variables in the CSP.
   - *classroom* - represents a classroom, which will be reserved by a course. It should have an attribute for the capacity, i.e., the number of students that the room can hold.
   - *reservation* - represents a reservation for a course. It should have attributes for the classroom and for the time slot and will be the value assigned to a course.
   - *constraints* can be represented in different ways. A general way is to have declarative forms for each constraint and have them converted to constraint objects. Consistency would be checked by mapping over the constraints and evaluating them as a conjunctive set on each assignment. A less general but simpler way is to represent them in a function that gets called on each assignment.

2. What data structure would you use as a variable and what would you use as a value?

   The variable would be the course data structure. The value would be the reservation data structure, which consists of a classroom and a time slot.

3. Describe a variable-ordering heuristic that could improve performance. This heuristic would be used in SELECT-UNASSIGNED-VARIABLE.

   There are multiple possible variable-ordering heuristics. One easy one is to select the course with the most students first. The idea behind this heuristic is to do the most constraining first.

4. Describe how you would compute the possible values for a variable in ORDER-DOMAIN-VALUES.

   The possible values for a course are all the unassigned reservations. That is, the possible values are all the classroom/time-slot pairings that have not been reserved, yet. A good heuristic to use is best fit, where fitness is defined as the least classroom capacity greater than the capacity required by the course.

5. Given a variable and a value assignment, describe how you would check that the assignment is consistent with the constraints. Note that this question may be straightforward, depending on how you generate the values.

   Depending on how you represented the constraints, consistency for an assignment is checked by applying either the declared constraints or the constraint function to the assignment. To reduce backtracking, one could do the constraint-checking at value generation time such that the only values searched are those that are valid.

6. How many representations of state do you need in your CSP solver? Remember that backtracking-search is a depth-first search that undoes its assignments when back tracking.

   Only one representation of state is required because inconsistent assignments are undone when back-tracking. That is, if a course assigned to a classroom and a time slot is found to be inconsistent, then the reservation is removed from the course.