CSE550: Simulated Dialogues Homework

In this homework, you will experiment with Simulated Dialogues, in which a computer system has a conversation with a simulated user.

Simulated dialogues are useful for a number of reasons. First, they allow researchers to experiment with what a suitable information state representation should be to allow a computer system to carry on interesting and natural conversations (Sidner 1994). Second, they allow researchers to investigate why certain language phenomena might exist, by studying the dialogue cost when a system and simulated user use the feature and do not use the feature (Walker 1995, Yang and Heeman 2004). Third, they allow researchers to apply reinforcement learning to automatically learn a dialogue policy for the system. This is our main purpose for simulating dialogues.

In this homework, we will have a computer system, specified using IS, and a simulated user, also specified using IS, talk to each other.


Question 1: Specification for System and User

In ISAgent3.tcl, you have update rules for the system for the banking domain. The rules are a simplified version of the ones used in the previous two homeworks. In particular, no agenda is used, nor are there any deliberate rules.

Writeup 1.1 Explain how the new rules allow the system to ask the user questions without an agenda or deliberate rules being used.

You also have update rules for the simulated user. The behavior of the simulated user is very straight-forward, as it will just answer the system’s questions. In fact, due to how simplistic the user is, the information state of the user does not include an agenda nor a QUD, as the user just uses LastMove to determine how to respond. Thus, an understanding rule is not even needed for the user.

Note that for more complex domains, we might need to specify the behavior of the simulated user with a number of rules, and we might even use some of the same rules for the simulated user and the system, such as for updating the QUD. However, for the banking application, we can keep things as simple as possible, and just use one rule.

Writeup 1.2 What IS variables are being used to describe B’s state? What code determines what B’s task is (how much does it want to transfer between which accounts).

Question 2: Change the IS Engine to Support a User Simulation

You will change the code in ISEngine3.tcl so that it will run a dialogue between the system and the simulated user. You will need to make a small change in the Run procedure, and more extensive changes to RunSide. The procedure RunSide is not well-named. What it does is run one cycle of the control strategy, running the appropriately understanding rules to understand what was just said, running any deliberation rules, and then running an action rule if the system has the turn.
You should change this code so that RunSide still does one cycle of the control strategy but for both the system (A) and the user (B). For RunSide, don’t bother passing it any parameters, you can just assume that the two agents are A and B. The first thing you want to do is figure out is who is the Speaker and who is the Hearer. Have the variable S indicate who the speaker is and H who the hearer is. Also, have isS be the information state of the speaker and isH be the information state of the hearer, which you can do with the following code:

```
upvar ::is$S isS
upvar ::is$H isH
```

After you make this change, you should be able to watch the system and simulated user have a dialogue.

**Writeup 2.3** Hand in a copy of the code for Run and RunSide and highlight what you changed.

**Question 3: Make a Simple System for Car Buying Domain**

For this question and the next, we return to the car-buying domain of the previous homework. In the next question, you will be make a simulated user that will interact with a system in order to create simulated dialogues.

In this question, you will simplify how the system works, so that the user simulations does not need to be overly complicated. You can either start with your own formulation of the system behavior or use the solution that was given in the answers for the previous homework.

First, we will assume that the user always has a goal of a car that is actually in the car database. Hence, you should remove support for the system to be able to re-ask the same question, as there will always be at least one car that matches the user’s answers.

Second, we will also assume that the user cares about all of the parameter values, and so you should remove support for ‘don’t care’.

As the changes to the system are very straight-forward, there is no reason to hand your specification.

**Question 4: Make a User Simulation for Car Buying Domain**

In this question, you will make a user simulation.

First, you need to give the user simulation a goal: a particular car that it wants. As mentioned in the previous question, we will assume that the user always has one of the cars in mind that is in the actual car database. Use an IS variable to hold the user’s goal.

Later in this homework, we will be even running multiple dialogue simulations in a row, and we want to make sure that each time the user has a different goal. The best way do to this is with the procedure InitAgent. This procedure is called by Run after all IS variables are given the initial values specified in SetupISVar. For setting the user’s goal, randomly pick one of the cars in the database.

Second, you need to specify an action rule to answer the system’s question. Do not worry about using a separate understanding and action selection rule. Simply do all of the processing in a single action selection rule, as we did in Question 1.

**Writeup 4.4** Hand in a copy of your user simulation. You should include any SetupISVar, InitAgent, and the update rule for the user. Where possible, use the same IS variables for both the user and the system.
Question 5: Costing a Dialogue

Add support so that you give a dialogue cost to a simulated dialogue. The cost of a dialogue can be viewed as the sum of the costs of the system’s actions that have been applied. We will add an IS variable to the system called Cost that will be used to the system’s action rules to indicate the cost of performing the action. The IS engine will use this variable to compute the total cost of the dialogue. Below are more detailed instructions.

**IS Engine:**  Change Run so that it initializes a global variable which will keep a tally of the action costs. Change RunSide so that after any action rule, it adds the cost of that was specified in the IS variable to the tally variable. Note that ISEngine should not know anything about how the costs should be set for the domain. That knowledge should just be in the agent file.

**Agent Specification:** Each utterance by the system (including the final bye action) should have a cost of 1.

For solution quality, the cost should be based on the final set of cars displayed to the user. If no cars have been displayed to the user, the cost should be 100. Due to how we have set up this problem, if any cars are displayed to the user, we are guaranteed that one of them will match (as we have the user pick its goal to be one of the cars in the database). But, you should charge 5 points for each extra car. So if 5 cars are displayed, one of them will be the solution. So 1 will be correct, and 4 of them will be wrong, giving a solution quality of 20. If only 1 car is displayed, it will be correct, and so the solution quality cost is 0. Note that none of the cars in hw5cars.tcl are the same, so you just need to see if the user has the same car in mind, rather than having to compare their slot value attributes.

Even though in the hand-crafted system update rules, the summary action is only applied once, immediately followed by the bye action, we want to prepare for the next homework assignment in which we relax this restriction. Hence, if the system happens to do more than one summary to the user, only the last one should count in computing the solution quality component of the cost function. Hence, you need to wait until the bye action (when you know the system is really finishing) to charge the solution quality cost. So, the summary action should save in an IS variable what cars it outputted to the user. Then, in the bye action, you should compare these cars with the user’s intended car, which you should have also saved in an IS variable. This logic should be in a subroutine that you call from the effects in the bye action. So the bye action will look like this:

```plaintext
SetupRule {A} action bye
AddEff {set is(NextMove) bye}
AddEff {set is(Cost) [finalcost]}
```

In the procedure finalcost, you can examine the value of A’s and B’s information state directly, using ::isA and ::isB. Note that finalcost should also take into account the dialogue cost of the bye action, in addition to the solution quality cost.

**Writeup 5.5** Hand in a copy of RunSide with the changes highlighted. Also had in a copy of your agent rules, highlighting what you changed.

Question 6: Running Multiple Dialogues

Add code so that Run is called 50 times. Have the code keep track of the cost of each dialogue run. Also, have the code compute a histogram of the dialogue costs (number of dialogue runs for each different cost in ascending order by cost), in a format similar to below:

```
5 3
```
Tcl’s associative arrays are very handy for this. You can use the command `array names` to determine all of the indices that you used, and use `lsort` to sort it in numerical ascending order.

Also, have the code compute the overall average. Report the average to two decimal places (which you can do using `[format %.2f $ave]`).

Writeup 6.6 Report the number of dialogue runs for each different cost and the average cost.

Question 7: Speeding up your code

In the last question, you ran 50 dialogue runs. In the next homework, you might be running a 1,000,000 dialogue simulations or more! So, let’s spend some time speeding up the code.

First, make sure all of the `puts` statements are inside of conditional statements that check if some debug flag is turned on. Do the same for any commands that update the tcl/tk window in Run and RunSide.

Second, for the system deliberate rule, which determines which cars match, you only need to call this if the system has the turn.

Third, you should speed up the code for `findcar`. This is actually one of the main reasons why the code is so slow, as the system needs to go through every car in the database for each system action. As we have removed the re-ask ability of the system, we actually know that as the dialogue processes, the matching cars is just a subset of the previous set of matching cars. So, we should just need to go over that list, rather than all of the cars.

With my laptop, I can process 100 dialogues in the random mode in under 10 seconds. You should have comparable results.

Writeup 7.7 Hand in your new deliberation rule, and the code for `findcars`. Explain anything else that you have changed to speed up `findcars`.

Writeup 7.8 Redo your previous question, but for 5000 dialogues. Is your average based on 5000 dialogues the same as for 50? If not, which do you think is closer to the real average?