CSCI-561 - Spring 2025 - Foundations of Artificial Intelligence

Homework 3

Deadline: Monday, April 21, 2025, 23:59:59 PST

Overview

In this assignment, you will explore **temporal reasoning** and use the **partially observable Markov decision process** (POMDP) to solve two problems. One is the "Little Prince" problem we introduced in the lecture, and the other involves speech recognition and text prediction. To solve the problems, you will need to:

- Execute a sequence of actions and make observations in a **POMDP** environment.
- Implement a temporal reasoning algorithm covered in this course.
- Return the most probable sequence of hidden states that the **POMDP** would traverse based on the actions, observations, and model.

Scenario 1: The Little Prince

The first scenario, depicted in Figure 1, is similar to the "Little Prince" from the lecture slides.

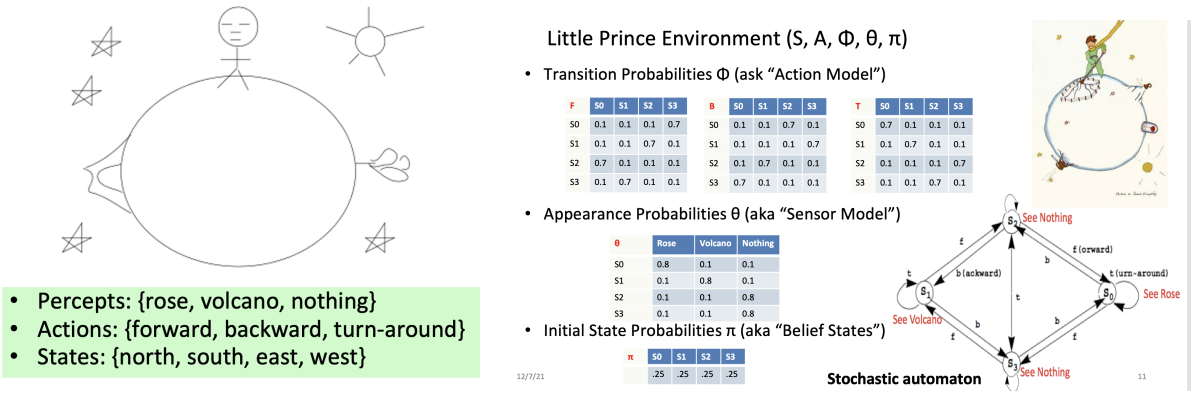

Figure 1: The Little Prince POMDP

You will be given a list of available percepts, actions, as well as states and the corresponding initial state weightages, transitions, and observation weight values in the environment. Your task is to design and implement a temporal-reasoning algorithm that takes a sequence of actions & observations and determines the most likely sequence of states that this POMDP has gone through, as shown in Figure 2.

For example, given the following actions & observations:

["rose", "forward", "none", …, "turn", …, "rose", "backward", "volcano"]

your program should return a sequence of hidden states such as the following:

["S2", "S3", …, "S2", "S3"]

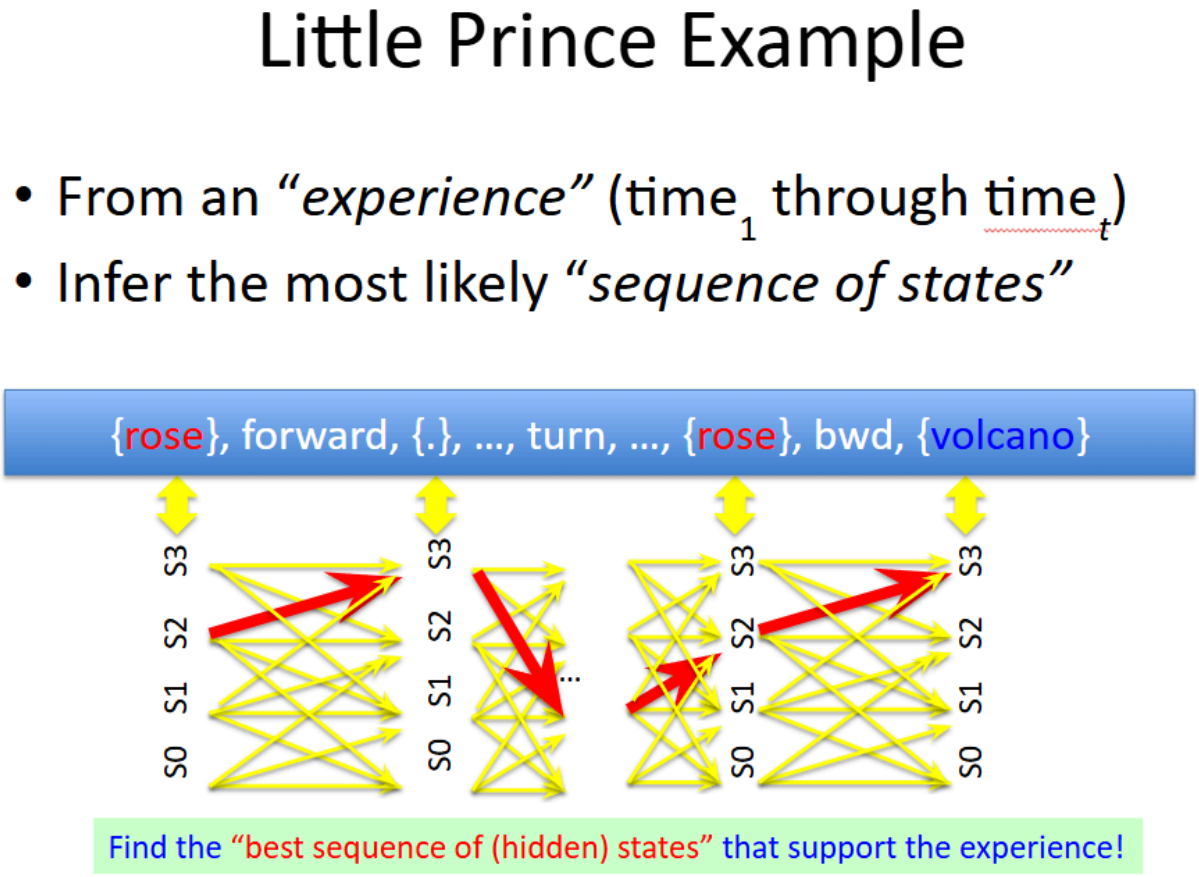

Figure 2: The inputs and outputs of a temporal-reasoning task

You can find more information about the solution in the slides and more information about the input data structure in the section "Input & Output Format" below.

Scenario 2: Speech Recognition

This scenario involves words, phonemes, and graphemes. A **phoneme** is the smallest phonetic unit in a language that distinguishes one word from another while a **grapheme** refers to the written symbols that correspond to the phoneme.

This model will primarily focus on distinguishing texts that share identical spoken utterances. For every word, you will be given a sequence of phonemes that represents the word, and you need to produce a grapheme sequence of the same length.

The following table provides some sample words and their phoneme/grapheme mappings:

Example Words	Phoneme Mapping	Grapheme Mapping
water	W | AO1 | T | ER0	w | a | t | er
human	Y | UW1 | M | AH0 | N	h | u | m | a | n

ocean	OW1 | SH | AH0 | N	o | c | ea | n

In the table, the phoneme corresponds with its graphemes for a given word. For example, the grapheme “a” in the word “water” corresponds to the phoneme “AO1” and the grapheme “er” corresponds to “ER0”.

NOTE: This task is based on prior work from the CMU Pronouncing Dictionary and uses a dialect of English known as North American English. This is why the word “human” is pronounced with a “Y” as the initial phoneme.

The ultimate goal of this task is to apply POMDP to finding the best sequence of text graphemes for a given sequence of phonemes. Specifically, the text graphemes correspond to states, and the phonemes correspond to observations. For convenience, we will use “N” (null) for actions. This ensures that we use a partially observable Markov **decision** process instead of just a partially observable Markov process. This also allows you to reuse code from the Little Prince scenario.

As part of the input, you will be provided with a dataset containing a list of grapheme-to-phoneme pairs along with a weight value for each pair. You will also be given grapheme-to-grapheme transition pairs with their weight values. The following example details the procedure for computing the probability tables. The weights correspond to un-normalized total probabilities P(o, s) and P(s, s’) (using the null action “N”), which were computed by counting (observation, state) and (state, state) pairs from a set of approximately 300k Wikipedia articles. Note that you will need to normalize these weights into the appropriate probability distributions P(o | s) and P(s’ | s, N).

Consider the following dataset:

Grapheme to Phoneme Mapping

Grapheme	Phoneme	Weight
s	S	100
s	Z	50
er	ER0	10
o	AH0	10
e	AH0	20

Grapheme to Grapheme Transition Mapping

Grapheme	Grapheme	Weight
s	er	80
s	o	10
s	e	10
er	o	5
o	e	8

Construction of Probability Tables :

- **Initial State Probability**: Computing the initial state / prior state distribution only involves dividing each weight in the state probability table by the total weight in the table.

  Given a state table as follows:

State	s	er	o	e

Weight	1	1	1	1

The initial probability P(s) is:

State	s	er	o	e
Prob	0.25	0.25	0.25	0.25

- **State Transition Probability**: This can be determined by looking at the weights of the transitions from one grapheme to another and calculating the probability through normalization for each (state, action) pair. This produces P(s|s,a). In this example, there is only a single valid action, so we don't show it in the table.

  The state transition probability for the sample dataset is:

States	s	er	o	e
s	0.0	0.8	0.1	0.1
er	0	0	1	0
o	0	0	0	1
e	0	0	0	0

- **Appearance Probability:** This can be inferred from the grapheme-phoneme pairs through normalization over all weights for a given state, producing the conditional distribution P(o|s). The appearance probability for the above example would be as follows:

	S	Z	ER0	AH0
s	0.667	0.333	0	0
er	0	0	1	0
o	0	0	0	1
e	0	0	0	1

Note how the row for state "s" sums to 1 over the different possible observations / phonemes "S" and "Z."

Your algorithm will be tested on a list of phonemes for which it should provide the most probable sequence of graphemes.

Input & Output Format

For each scenario, you will be given a set of input files that contain the table of weights as described above. You need to parse and normalize them into the appropriate conditional probabilities.

These files contain:

- An **indicator** in the first line that denotes what data is stored in it.
- The **number of entries** and the **default weight**.
- A sequence of entries where states, observations, and actions are wrapped in double quotes, and the weights are specified as integers.
- **An empty new line** at the end of the file.

There are three types of weight table files:

The first contains weights for every state and describes the prior probability of each – P(s). The default weight in this file is only for ensuring format consistency across all types of files. Thus, you don't need to use it.

state_weights.txt:

```
state_weights
<number of states> <default weight>
"state1" <weight of state1>
"state2" <weight of state2>
etc…
```

The second file contains weights for (state, action, state) triples and describes the probability of state transitions – P(s, a, s). Triples not specified in the table should be given the default weight from the second line. Note that you need to normalize these weights into the appropriate probability distribution P(s' | a, s).

state_action_state_weights.txt:

```
state_action_state_weights
<number of triples in file> <number of unique states> <number of unique
actions> <default weight>
"state1" "action1" "next state1" <weight of (state1, action1, next state1)>
"state2" "action2" "next state2" <weight of (state2, action2, next state2)>
etc…
```

The third of these files contains weights for every (state, observation) pair, and describes the probability of each state observation pair P(s, o). Pairs not specified in the table should be given the default weight specified in the second line. Note that you will need to normalize these weights into the appropriate probability distribution P(o | s).

state_observation_weights.txt:

```
state_observation_weights
<number of pairs in file> <number of unique states> <number of unique
observations> <default weight>
“state1” “observation1” <weight of (state1, observation1)>
“state2” “observation2” <weight of (state1, observation1)>
etc…
```

Lastly, you will receive a file containing the sequence of (observation, action) pairs, on which you should run the Viterbi algorithm.

observation_actions.txt:

```
observation_actions
<number of pairs in file>
“observation1” “action1”
“observation2” “action2”
etc…
```

Your code should produce a file containing the predicted state sequence.

states.txt:

```
states
<length of state sequence>
“state1”
“state2”
etc…
```

Sample Test Case

The following sample test case corresponds to the Little Prince scenario (the format matches that used for the speech recognition scenario)

Little Prince scenario test case:

state_weights.txt

```
state_weights
3 0
"S0" 2
"S1" 5
"S2" 5
```

state_observation_weights.txt

```
state_observation_weights
9 3 3 0
"S0" "Volcano" 3
"S0" "Grass" 3
"S0" "Apple" 2
"S1" "Volcano" 5
"S1" "Grass" 5
"S1" "Apple" 2
"S2" "Volcano" 3
"S2" "Grass" 5
"S2" "Apple" 2
```

state_action_state_weights.txt

```
state_action_state_weights
27 3 3 0
"S0" "Forward" "S0" 3
"S0" "Forward" "S1" 3
"S0" "Forward" "S2" 2
"S1" "Forward" "S0" 4
"S1" "Forward" "S1" 5
"S1" "Forward" "S2" 1
"S2" "Forward" "S0" 1
"S2" "Forward" "S1" 5
"S2" "Forward" "S2" 4
"S0" "Backward" "S0" 5
"S0" "Backward" "S1" 5
"S0" "Backward" "S2" 5
"S1" "Backward" "S0" 3
"S1" "Backward" "S1" 4
"S1" "Backward" "S2" 1
"S2" "Backward" "S0" 2
"S2" "Backward" "S1" 3
"S2" "Backward" "S2" 3
"S0" "Turnaround" "S0" 5
"S0" "Turnaround" "S1" 3
"S0" "Turnaround" "S2" 5
"S1" "Turnaround" "S0" 3
"S1" "Turnaround" "S1" 4
"S1" "Turnaround" "S2" 2
"S2" "Turnaround" "S0" 1
"S2" "Turnaround" "S1" 2
"S2" "Turnaround" "S2" 2
```

observation_actions.txt

```
observation_actions
4
"Apple" "Turnaround"
"Apple" "Backward"
"Apple" "Forward"
"Volcano"
```

Output:

states.txt

```
states
4
"S2"
"S2"
"S2"
"S1"
```

Input Constraints & Time Limits

Little Prince:

Maximum Number of States in the Environment	10 states
Maximum Number of Percepts in the Environment	10 percepts
Number of Actions Possible in the Environment (Fixed)	3 (“Forward”, “Backward”, “Turnaround”)
Maximum Length of Observation Action Sequence	20
Time Limit Allowed per Test Case	1 second
Number of Test Cases	20 (5 preliminary, 15 hidden)

Speech Recognition:

Maximum Number of States in the Environment	682
Maximum Number of Observations in the Environment	69
Number of Actions Possible in the Environment (Fixed)	“N” (Indicates null action as in the scenario description)
Maximum Length of Observation Action Sequence	100 steps
Time Limit allowed per test case	1 minute
Number of test cases	30 (5 Preliminary and 25 hidden)

Grading Criteria

Submissions before the deadline will be evaluated on sample test cases. After that, your final submission will be evaluated on the grading set. The grading set includes 50 test cases (20 for the Little Prince and 30 for speech recognition). Each test case is worth 2 points.

The performance of your program will be computed automatically by comparing your output sequence with the most likely known sequence, and the matching percentage will determine the grade of your submissions. More specifically, the probability of the hidden state sequence that your solution generated (p1) will be compared with the probability of the hidden state sequence proposed by the TA's agent (p2), and p1/ p2 will be used as the final score for each test case.

$$\text{Score for a single test case} = 2 * \frac{\text{Probability of student's state sequence}}{\text{Probability of TA agent's state sequence}}$$

Final Score = Sum of scores across all test cases

NOTE : The max score for a single test case will be capped at 2 points.

The probability of the state sequence will be calculated based on the joint probability of the state sequence, along with the action sequence and the observation sequence.

In the following example, the score is calculated as follows:

State List = [S0, S1, S2], Actions Set= [N], Observations Set= [A, B]

Initial State Probability

S0	S1	S2
0.5	0.25	0.25

Appearance Probability

	A	B
S0	0.9	0.1
S1	0.2	0.8
S2	0.5	0.5

Transition Probability

	S0	S1	S2
S0	0.5	0.3	0.2
S1	0.9	0.1	0
S2	0.2	0.8	0

Observation Action Sequence = [A, <N>, B,<N>, B]

```
If the student's predicted state sequence is [S0, S1, S1], then
Student's Probability = π(S0)*P(A|S0)*P(S1|S0,<N>)*P(B|S1)*P(S1|S1,<N>)*P(B|S1)
                      = 0.5*0.9*0.3*0.8*0.1*0.8
                      = 0.00864

If the TA's predicted state sequence is [S0, S1, S0], then
TA's Probability = π(S0)*P(A|S0)*P(S1|S0,<N>)*P(B|S1)*P(S0|S1,<N>)*P(B|S0)
                 = 0.5 * 0.9 * 0.3 *0.8 * 0.9 * 0.1
                 = 0.00972

Your score for this test case = 2 * 0.00864 / 0.00972  = 1.778
```

Notes

- Please name your program "**my_solution.xxx**", where 'xxx' is the extension for the programming language you choose ("py" for Python, "cpp" for C++, and "java" for Java). If you are using C++11, the name of your file should be "my_solution11.cpp" and if you are using python3, the name of your file should be "my_solution3.py". Please use only the programming languages mentioned above for this homework. Please note the newest version of Python that is offered is Python 3.7.5; hence, the walrus operator and other features in newer versions of Python are not supported.

- The time limit is the total combined CPU time as measured by the Unix **time** command. This command measures the pure computation time used by your program and discards the time taken by the operating system, disk I/O, program loading, etc. Beware that it accumulates time spent in any threads spawned by your agent (so if you run 4 threads and use 400% CPU for 10 seconds, this will count as 40 seconds of allocated time). Your local machine may be more powerful and run your code faster, but remember that your code will be evaluated on Vocareum.

- **Please don't copy code from any website as our plagiarism agent will certainly detect it. This includes Wikipedia.**

- There may be a lot of Q&A on Piazza. **Please always search for relevant questions before posting a new one. Duplicate questions make everyone's life harder.**

- **Only submit the source code files (in .java, .py or .cpp). All other files should be excluded.**

- Please submit your homework code through Vocareum (https://labs.vocareum.com/) under the assignment HW3. Your username is your email address. Click "forgot password" for the first time login. You should have been enrolled in this course on Vocareum. If not, please post a private question with your email address and USC ID on Piazza so that we can invite you again.

- You can submit your homework code (by clicking the "submit" button on Vocareum) as many times as you want. Only the latest submission will be considered for grading. After the initial deadline, the submission window will still be open for 5 days. However, a 20% per day penalty will be applied if your last submission is after the deadline.

- Every time you click the "submit" button, you can view your submission report to see if your code works. The grade report will be released after the due date of the project.

- You don't have to keep the page open on Vocareum while the scripts are running.

- **Since you CANNOT delete old submissions, AVOID multiple submissions of large files to Vocareum**. If you run out of space, we will have to contact Vocareum support to delete them.