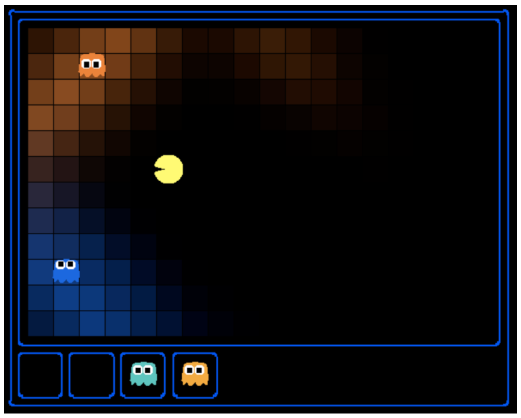
Graphical models / factor graphs bonus project

Due: March 25, 11:59pm
Late policy: 3%/day, see course webpage for details



I can hear you, ghost.

Running won't save you from my

Particle filter!

Introduction

Pacman spends his life running from ghosts, but things were not always so. Legend has it that many years ago, Pacman's great grandfather Grandpac learned to hunt ghosts for sport. However, he was blinded by his power and could only track ghosts by their banging and clanging.

In this project, you will design Pacman agents that use sensors to locate and eat invisible ghosts. You'll advance from locating single, stationary ghosts to hunting packs of multiple moving ghosts with ruthless efficiency.

The code for this project contains the following files, available as a zip archive.

Files you'll edit:

`inference.py`	Code for tracking ghosts over time using their sounds.
`factorOperations.py`	Operations to compute new joint or magrinalized probability tables.

Files you will not edit:

`busters.py` The main entry to Ghostbusters (replacing Pacman.py)

`bustersGhostAgents.py` New ghost agents for Ghostbusters

`distanceCalculator.py` Computes maze distances, caches results to avoid re-computing.

`game.py` Inner workings and helper classes for Pacman

`ghostAgents.py` Agents to control ghosts

`graphicsDisplay.py` Graphics for Pacman

`graphicsUtils.py` Support for Pacman graphics

`keyboardAgents.py` Keyboard interfaces to control Pacman

`layout.py` Code for reading layout files and storing their contents

`util.py` Utility functions

Files to Edit and Submit: You will fill in portions of `factorOperations.py` and `inference.py` during the assignment. Please *do not* change the other files in this distribution. Upload both of those files to gradescope.

Software environment: See Ed Discussions pinned post at the top titled 'Python environment/dependencies'. The autograding server, which decides on your actual grade, runs python 3.12, so you are strongly encouraged to use that version to ensure compatibility. Technically, you can still run the code using a different python version, but take care to double check that the autograding server on gradescope is reporting the same final grades as what you see on your local machine.

Evaluation: Your code will be autograded for correctness. Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. You should regard the Gradescope autograder's judgments *on the server* as your final grade, *not* what you see locally on your laptop (though ideally these agree). Remember that you can submit as many times to the server as you want before the deadline in order to make sure that the code works correctly on the grading server. In unusual extenuating circumstances we will manually run your code, for example if we suspect grade-server hacking, but by strong default, what the server says is your final grade. In unusual extenuating circumstances we will manually run your code, for example if we suspect grade-server hacking, but by strong default, what the server says is your final grade.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours and the discussion forum are there for your support; please use them. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Ghostbusters and Bayes Nets

In the CS 4700 version of Ghostbusters, the goal is to hunt down scared but invisible ghosts. Pacman, ever resourceful, is equipped with sonar (ears) that provides noisy readings of the Manhattan distance to each ghost. The game ends when Pacman has eaten all the ghosts. To start, try playing a game yourself using the keyboard.

```
python busters.py
```

The blocks of color indicate where the each ghost could possibly be, given the noisy distance readings provided to Pacman. The noisy distances at the bottom of the display are always non-negative, and always within 7 of the true distance. The probability of a distance reading decreases exponentially with its difference from the true distance.

Your primary task in this project is to implement inference to track the ghosts. For the keyboard based game above, a crude form of inference was implemented for you by default: all squares in which a ghost could possibly be are shaded by the color of the ghost. Naturally, we want a better estimate of the ghost's position. Fortunately, Bayes Nets provide us with powerful tools for making the most of the information we have. Throughout the rest of this project, you will implement algorithms for performing both exact and approximate inference using Bayes Nets. The project is challenging, so we do encouarge you to start early and seek help when necessary.

While watching and debugging your code with the autograder, it will be helpful to have some understanding of what the autograder is doing. There are 2 types of tests in this project, as differentiated by their `.test` files found in the subdirectories of the `test_cases` folder. For tests of class `DoubleInferenceAgentTest`, you will see visualizations of the inference distributions generated by your code, but all Pacman actions will be pre-selected according to the actions of the staff implementation. This is necessary to allow comparision of your distributions with the staff's distributions. The second type of test is `GameScoreTest`, in which your `BustersAgent` will actually select actions for Pacman and you will watch your Pacman play and win games.

As you implement and debug your code, you may find it useful to run a single test at a time. In order to do this you will need to use the `-t` flag with the autograder. For example if you only want to run the first test of question 1, use:

```
python autograder.py -t test_cases/q1/1-ObsProb
```

In general, all test cases can be found inside `test_cases/q*`.

For this project, it is possible sometimes for the autograder to time out if running the tests with graphics. To accurately determine whether or not your code is efficient enough, you should run the tests with the `--no-graphics` flag. If the autograder passes with this flag, then you will receive full points, even if the autograder times out with graphics.

Bayes Nets and Factors

First, take a look at `bayesNet.py` to see the classes you'll be working with - `BayesNet` and `Factor`. You can also run this file to see an example `BayesNet` and associated `Factors`: `python bayesNet.py`

You should look at the `printStarterBayesNet` function - there are helpful comments that can make your life *much* easier later on.

The Bayes Net created in this function is shown below:

(Raining --> Traffic <-- Ballgame)

A summary of the terminology is given below:

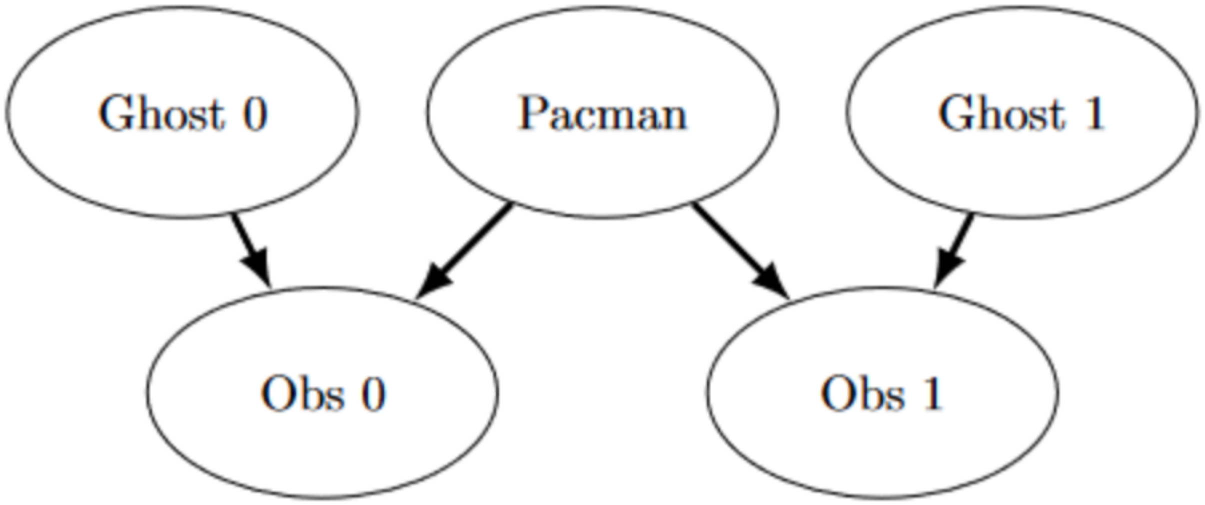
- `Bayes Net`: This is a representation of a probabilistic model as a directed acyclic graph and a set of conditional probability tables, one for each variable, as shown in lecture. The Traffic Bayes Net above is an example.
- `Factor`: This stores a table of probabilities, although the sum of the entries in the table is not necessarily 1. A factor is of the general form $f(X_1, ...X_m, y_1, ..., y_n|Z_1, ..., Z_p, w_1, ..., w_q)$. Recall that lower case variables have already been assigned. For each possible assignment of values to the X_i and Z_j variables, the factor stores a single number. The Z_j, w_k variables are said to be conditioned while the X_i, y_l variables are unconditioned.
- Conditional Probability Table (CPT): This is a factor satisfying two properties:
    1. Its entries must sum to 1 for each assignment of the conditional variables
    2. There is exactly one unconditioned variable

The Traffic Bayes Net stores the following CPTs: $P(Raining), P(Ballgame), P(Traffic|Ballgame, Raining)$.

Question 1 (2 points): Bayes Net Structure

Implement the `constructBayesNet` function in `inference.py`. It constructs an empty Bayes net with the structure described below. A Bayes Net is incomplete without the actual probabilities, but factors are defined and assigned by staff code separately; you don't need to worry about it. If you are curious, you can take a look at an example of how it works in `printStarterBayesNet` in `bayesNet.py`. Reading this function can also be helpful for doing this question.

The simplified ghost hunting world is generated according to the following Bayes net:



Don't worry if this looks complicated! We'll take it step by step. As described in the code for `constructBayesNet`, we build the empty structure by listing all of the variables, their values, and the edges between them. This figure shows the variables and the edges, but what about their values?

- Add variables and edges based on the diagram.
- Pacman and the two ghosts can be anywhere in the grid (we ignore walls for this). Add all possible position tuples for these.
- Observations here are Manhattan distances of Pacman to ghosts $\pm$ noise, and are non-negative.

Grading: To test and debug your code, run

```
python autograder.py -q q1
```

Question 2 (4 points): Join Factors

Implement the `joinFactors` function in `factorOperations.py`. It takes in a list of `Factors` and returns a new `Factor` whose probability entries are the product of the corresponding rows of the input `Factors`.

`joinFactors` can be used as the product rule, for example, if we have a factor of the form $P(X|Y)$ and another factor of the form $P(Y)$, then joining these factors will yield $P(X,Y)$. So, `joinFactors` allows us to incorporate probabilities for conditioned variables (in this case, Y). However, you should not assume that `joinFactors` is called on probability tables -- it is possible to call `joinFactors` on `Factors`whose rows do not sum to 1.

Grading: To test and debug your code, run

```
python autograder.py -q q2
```

It may be useful to run specific tests during debugging, to see only one set of factors print out. For example, to only run the first test, run:

```
python autograder.py -t test_cases/q2/1-product-rule
```

Hints and Observations:

- Your `joinFactors` should return a *new* `Factor`.
- Here are some examples of what `joinFactors` can do:
  - joinFactors$(P(X|Y), P(Y)) = P(X, Y)$
  - joinFactors$(P(V, W|X, Y, Z), P(X, Y|Z)) = P(V, W, X, Y|Z)$
  - joinFactors$(P(X|Y, Z), P(Y)) = P(X, Y|Z)$
  - joinFactors$(P(V|W), P(X|Y), P(Z)) = P(V, X, Z|W, Y)$
- For a general `joinFactors` operation, which variables are unconditioned in the returned Factor? Which variables are conditioned?
- Factors store a `variableDomainsDict`, which maps each variable to a list of values that it can take on (its domain). A `Factor` gets its `variableDomainsDict` from the `BayesNet` from which it was instantiated. As a result, it contains all the variables of the `BayesNet`, *not* only the unconditioned and conditioned variables used in the `Factor`. For this problem, you may assume that all the input `Factor`s have come from the same `BayesNet`, and so their `variableDomainsDicts` are all the same.

Question 3 (4 points): Eliminate (not ghosts yet)

Implement the `eliminate` function in `factorOperations.py`. It takes a `Factor` and a variable to eliminate and returns a new `Factor` that does not contain that variable. This corresponds to summing all of the entries in the `Factor` which only differ in the value of the variable being eliminated.

Grading: To test and debug your code, run

```
python autograder.py -q q3
```

It may be useful to run specific tests during debugging, to see only one set of factors print out. For example, to only run the first test, run:

```
python autograder.py -t test_cases/q3/1-simple-eliminate
```

Hints and Observations:

- Your `eliminate` should return a *new* `Factor`.
- `eliminate` can be used to marginalize variables from probability tables. For example:
  - eliminate$(P(X, Y|Z), Y) = P(X|Z)$
  - eliminate$(P(X, Y|Z), X) = P(Y|Z)$
- For a general eliminate operation, which variables are unconditioned in the returned Factor? Which variables are conditioned?
- Remember that `Factor`s store the `variableDomainsDict` of the original `BayesNet`, and *not* only the unconditioned and conditioned variables that they use. As a result, the returned `Factor` should have the same `variableDomainsDict` as the input `Factor`.

Question 4 (3 points): Variable Elimination

Implement the `inferenceByVariableElimination` function in `inference.py`. It answers a probabilistic query, which is represented using a `BayesNet`, a list of query variables, and the evidence.

Grading: To test and debug your code, run

```
python autograder.py -q q4
```

It may be useful to run specific tests during debugging, to see only one set of factors print out. For example, to only run the first test, run:

```
python autograder.py -t test_cases/q4/1-disconnected-eliminate
```

Hints and Observations:

- The algorithm should iterate over hidden variables in elimination order, performing joining over and eliminating that variable, until the only the query and evidence variables remain.
- The sum of the probabilities in your output factor should sum to 1 (so that it is a true conditional probability, conditioned on the evidence).
- Look at the `inferenceByEnumeration` function in inference.py for an example on how to use the desired functions. (Reminder: Inference by enumeration first joins over all the variables and then eliminates all the hidden variables. In contrast, variable elimination interleaves join and eliminate by iterating over all the hidden variables and perform a join and eliminate on a single hidden variable before moving on to the next hidden variable.)
- You will need to take care of the special case where a factor you have joined only has one unconditioned variable (the docstring specifies what to do in greater detail).

Submission

Submit `factorOperations.py` and `inference.py` to Gradescope.

Please specify in a comment at the top of your python files any partner you may have worked with and verify that both you and your partner are associated with the submission after submitting.

Acknowledgements. This project is taken from Berkeley CS188, whose course materials are available online at `ai.berkley.edu`.