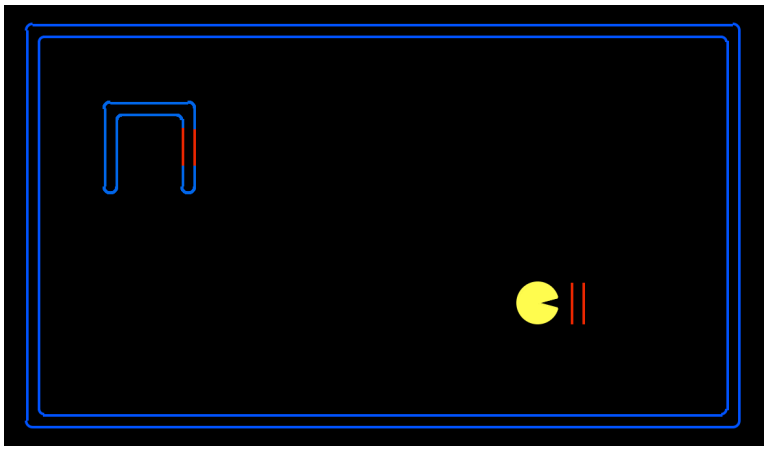
Universidad
Rey Juan Carlos

Práctica 3: Redes Bayesianas[1]

Inteligencia Artificial



Introducción

En esta práctica, se implementarán algoritmos de inferencia para redes Bayesianas, específicamente eliminación de variables y computaciones de valor de información perfecta. Esos algoritmos de inferencia permitirán razonar la existencia de píldoras y fantasmas.

Se puede ejecutar el autoevaluador para un test particular con comandos de la forma:

```
python3 autograder.py -t test_cases/q4/1-simple-eliminate
```

El código de este proyecto está disponible en el Aula Virtual para descarga y contiene los siguientes ficheros:

Archivos a editar	
`factorOperations.py`	Operaciones sobre Factores (unión, eliminación, normalización).

[1] Esta práctica es una adaptación de una práctica de la Asignatura de Inteligencia Artificial de la Universidad Carnegie Mellon, disponible aquí.

`inference.py`	Algoritmos de inferencia (enumeración, eliminación de variables, ponderación de probabilidad).
`bayesAgents.py`	Agentes Pacman que razonan bajo incertidumbre.
Archivos que deberían consultarse	
`bayesNet.py`	Las clases BayesNet y Factor.
Archivos que pueden ignorarse	
`graphicsDisplay.py`	Parte gráfica de Pacman.
`graphicsUtils.py`	Soporte para la parte gráfica de Pacman.
`textDisplay.py`	Gráficos ASCII para Pacman.
`ghostAgents.py`	Agentes para el control de los fantasmas.
`keyboardAgents.py`	Interfaz de teclado para el control de Pacman.
`layout.py`	Código para leer los archivos de layout y guardar su contenido.
`autograder.py`	Autoevaluador del proyecto.
`testClasses.py`	Clases generales de test para la autoevaluación.
`test_cases/`	Directorio que contiene los casos de test para cada pregunta.
`bayesNets2TestClasses.py`	Clases de test para la autoevaluación propias de la práctica.

Pacman caza tesoros

Pacman ha entrado en un mundo de misterio. Inicialmente, el mapa completo es invisible. Según explora el entorno va aprendiendo información de las casillas adyacentes. Este mapa contiene 2 casas: una casa fantasma, que es en su mayoría roja y una casa de comida, que es en su mayoría azul. El objetivo que persigue Pacman es entrar en la casa de comida mientras evita la casa fantasma.

Pacman irá razonando qué casa es cada una según las observaciones que vaya realizando y razonando con un límite de confianza para escoger entre realizar una elección de casa o recoger más pruebas. Para conseguir esto, se implementará inferencia probabilística utilizando redes Bayesianas.

Para probar el juego:

```
python3 hunters.py -p KeyboardAgent -r
```

Redes Bayesianas y Factores

En primer lugar, se puede revisar el archivo `bayesNet.py` para ver las clases con las que se trabajará: `BayesNet` y `Factor`. Se puede ejecutar este archivo para ver un ejemplo de `BayesNet` y los `Factor` asociados: `python3 bayesNet.py`

Se debería revisar la función `printStarterBayesNet` que tiene comentarios que podrían hacer que se entendiese más rápido el funcionamiento.

Las redes Bayesianas creadas en esta función siguen la siguiente forma:

(Raining → Traffic ← Ballgame)

Resumiendo la terminología:

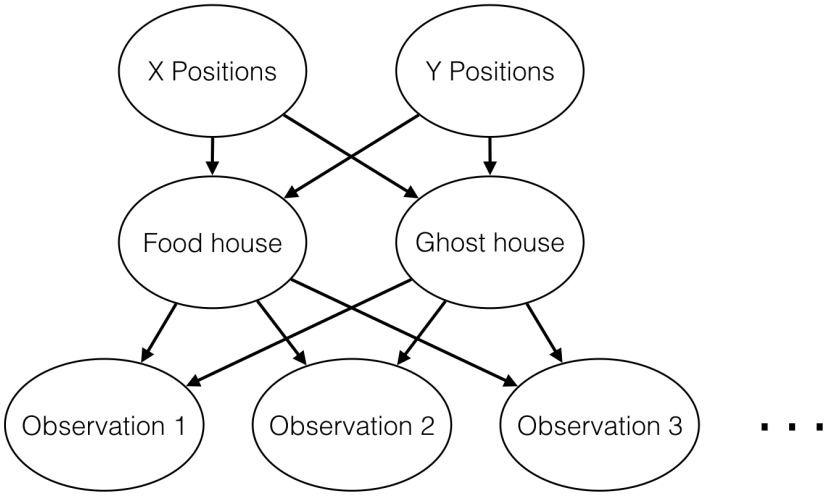
- `BayesNet`: esta es una representación de un modelo probabilístico como un grafo acíclico dirigido y una serie de tablas de probabilidad condicional, una por cada variable.
- `Factor`: guarda una tabla de probabilidades, aunque la suma de las entradas en la tabla no tiene que ser necesariamente 1. Un factor sigue la forma general $P(X_1,...,X_m, y_q,...,y_n|Z_1,...,Z_p, w_1,...,w_q)$. Las variables en minúscula ya han sido asignadas. Para cada posible asignación de valores a las variables X_i y Z_j, el factor guarda un único número. Las variables Z_j, w_k son condicionadas mientras que las variables X_i, y_l son no condicionadas.
- La tabla de probabilidad condicional (CPT): es un factor que satisface dos propiedades:
    1. Sus entradas tienen que sumar 1 para cada asignación de las variables condicionadas.
    2. Hay exactamente 1 variable no condicionada.

La red Bayesiana de tráfico guarda las siguientes CPTs: *P(Raining), P(Ballgame), P(Traffic|Ballgame, Raining).*

Pregunta 1 (3 puntos): estructura de la red Bayesiana

Hay que implementar la función `constructBayesNet` en `BayesAgents.py`. Esta construye una red Bayesiana vacía con la estructura descrita debajo.

El mundo de la caza del tesoro se genera según la siguiente red Bayesiana:



Según se describe en el código de `constructBayesNet`, se construye la estructura vacía enumerando todas las variables, sus valores y las aristas entre ellas. La figura muestra las variables y las aristas pero, ¿dónde están los valores?

- Las *X Positions* determinan qué casa va en qué parte del tablero. Puede ser *food-left* o *ghost-left*.
- Las *Y Positions* determinan cómo se orientan las casas verticalmente. Estas modelan las posiciones verticales de ambas casas a la vez y toma uno de los siguientes valores: *both-top*, *both-bottom*, *left-top* o *left-bottom*. El *left-top*: la casa en la parte izquierda del tablero está en la parte superior y la casa en la parte derecha del tablero en la parte inferior.
- La *Food house* y *Ghost house* especifican las posiciones reales de las dos casas. Ambas son funciones determinísticas de "*X positions*" y "Y Positions".
- Las observaciones son mediciones que Pacman hace mientras viaja por el tablero. Hay que tener en cuenta que hay muchos de estos nodos, uno por cada posición del tablero que podría ser la pared o la casa. Si no hay casa en una posición concreta, la observación correspondiente es nula y de otro modo es roja o azul, con la distribución precisa de colores dependiendo del tipo de casa.

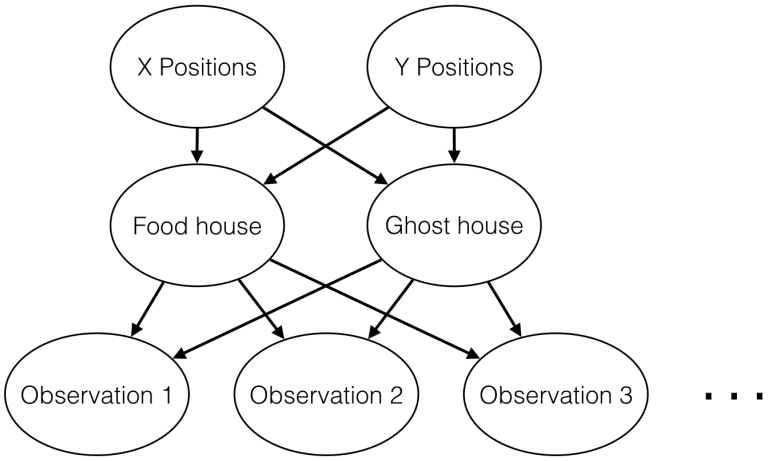
Para comprobar el código:

```
python3 autograder.py -q q1
```

Pregunta 2 (1 puntos): probabilidades de la red Bayesiana

Se pide implementar la función `fillYCPT` en `bayesAgents.py`. Este toma la red Bayesiana que se ha construido previamente en la pregunta anterior y especifica los factores que gobiernan las variables *Y position* (Ya se han completado los factores de X *position* , *house* y *observation*).

La estructura de la red Bayesiana sería, de nuevo:



Para ver un ejemplo de cómo construir factores, se puede ver la implementación del factor para las *X positions* en `fillXCPT`.

Las *Y positions* vienen dadas por los valores `BOTH_TOP_VAL`, `BOTH_BOTTOM_VAL`, `LEFT_TOP_VAL`, `LEFT_BOTTOM_VAL`. Estas variables y sus probabilidades asociadas `PROB_BOTH_TOP`, `PROB_BOTH_BOTTOM`, `PROB_ONLY_LEFT_TOP`, `PROB_ONLY_LEFT_BOTTOM` vienen dadas por las constantes en la parte superior del documento.

Pista: solo hay cuatro entradas en el factor Y *position*, así que se pueden introducir a mano.

Para comprobar el código:

```
python3 autograder.py -q q2
```

Pregunta 3 (5 puntos): unir factores

Hay que implementar la función `joinFactors` en `factorOperations.py`. Esta recibe una lista de `Factor`s y devuelve un nuevo `Factor` cuyas entradas de probabilidad son el producto de las correspondientes filas de los `Factor`s de entrada.

Los `joinFactors` pueden ser utilizados por la regla del producto, por ejemplo si se tiene el factor de la forma $P(X|Y)$ y otro factor de la forma $P(Y)$ entonces uniendo estos factores se producirá $P(X, Y)$. Entonces `joinFactors` permite incorporar probabilidades de variables condicionadas (en este caso Y). Sin embargo, no se debería asumir que `joinFactors` se llama en las tablas de probabilidad, es posible llamar a `joinFactors` en `Factors` cuyas filas no sumen 1.

Para comprobar el código:

```
python3 autograder.py -q q3
```

Podría ser útil utilizar alguno de los tests específicos durante la fase de depuración para ver solo un conjunto de factores impreso. Por ejemplo, para ejecutar el primer test:

```
python3 autograder.py -t test_cases/q3/1-product-rule
```

Pistas y observaciones:

- Los `joinFactors` deberían devolver un nuevo `Factor`.
- Algunos ejemplos de lo que `joinFactors` puede hacer:
  - *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)*
- Para una operación `joinFactors` general, ¿qué variables son incondicionadas en el *Factor* devuelto?, ¿qué variables están condicionadas?
- Los *Factors* guardan `variableDomainsDict`, que mapean cada variable a una lista de valores que pueden recibir (su dominio). Un `Factor` coge su `variableDomainDict` de la `BayesNet` desde la cual ha sido instanciada. Como resultado, contiene todas las variables de `BayesNet`, no solo las variables no condicionadas y las condicionadas usadas en el `Factor`. Para este problema, se puede asumir que todos los `Factor` de entrada vienen del mismo `BayesNet` y por tanto sus `variableDomainDicts` son todos iguales.

Pregunta 4 (4 puntos): eliminación

Se pide implementar la función `eliminate` en `factorOperations.py`. Esta recibe un `Factor` y una variable para eliminar y devuelve un nuevo `Factor` que no contiene dicha variable. Esto

corresponde a sumar todas las entradas en el `Factor` que solo difieren en el valor de la variable siendo eliminada.

Para comprobar el código:

```
python3 autograder.py -q q4
```

Podría ser útil utilizar alguno de los tests específicos durante la fase de depuración para ver solo un conjunto de factores impreso. Por ejemplo, para ejecutar el primer test:

```
python3 autograder.py -t test_cases/q4/1-simple-eliminate
```

Pistas y observaciones:

- La función `eliminate` debería devolver un nuevo `Factor`.
- La función `eliminate` puede ser utilizada para marginalizar variables desde las tablas de probabilidad. Por ejemplo:
    - eliminate(P(X,Y|Z),Y) = P(X|Z)
    - eliminate(P(X,Y|Z), X) = P(Y|Z)
- Para una operación general de eliminación, ¿qué variables son incondicionadas en el Factor devuelto? ¿Qué variables son condicionadas?
- Hay que recordar que los `Factor` guardan el `variableDomainDict` de la `BayesNet` original y no solo las variables incodicionadas y condicionadas que utiliza. Por lo tanto, el `Factor` devuelto debería tener el mismo `variableDomainDict` que el `Factor` de entrada

Pregunta 5 (4 puntos): normalización

Hay que implementar la función `normalize` en `factorOperations.py`. Esta recibe un `Factor` como entrada y lo normaliza, esto es, escala todas las entradas en el `Factor` para que la suma de las entradas en el `Factor` sea 1. Si la suma de las probabilidades en el `Factor` de entrada es 0, se debería devolver `None`.

Para comprobar el código:

```
python3 autograder.py -q q5
```

Podría ser útil utilizar alguno de los tests específicos durante la fase de depuración para ver solo un conjunto de factores impreso. Por ejemplo, para ejecutar el primer test:

```
python3 autograder.py -t test_cases/q5/1-preNormalized
```

Pistas y observaciones:

- La función `normalize` debería devolver un nuevo `Factor`.
- La función `normalize` no afecta a las distribuciones de probabilidad (porque estas ya deben sumar 1).

- Para una operación `normalize` general, ¿qué variables son no condicionadas en el `Factor` devuelto?, ¿qué variables son condicionadas? Hay que asegurarse de leer el docstring de `normalize` para tener más instrucciones. En particular, el tratamiento de las variables no condicionadas con exactamente 1 entrada en el dominio.
- Hay que recordar que `Factors` guarda el `variableDomainsDict` de la `BayesNet` original y no solo de las variables condicionadas e incondicionadas que usa. Por lo tanto, el `Factor` devuelto debería tener el mismo `variableDomainsDict` que el `Factor` de entrada.

Pregunta 6 (4 puntos): eliminación de variables

Se pide implementar la función `inferenceByVariableElimination` en `inference.py`. Esta responde a una consulta probabilística, que es representada utilizando una `BayesNet`, una lista de variables de consulta y una evidencia.

Para comprobar el código:

```
python3 autograder.py -q q6
```

Podría ser útil utilizar alguno de los tests específicos durante la fase de depuración para ver solo un conjunto de factores impreso. Por ejemplo, para ejecutar el primer test:

```
python3 autograder.py -t test_cases/q6/1-disconnected-eliminate
```

Pistas y observaciones:

- El algoritmo debería iterar sobre las variables ocultas en orden de eliminación, ejecutando una unión y eliminando dicha variable hasta que solo queden la consulta y la variable de evidencia.
- La suma de probabilidades en el factor de salida debería sumar 1 para que sea una probabilidad condicional verdadera, condicionada a la evidencia.
- Se puede echar un vistazo a la función `inferenceByEnumeration` dentro de `inference.py` para un ejemplo de cómo usar las funciones deseadas. Se recuerda que la inferencia por enumeración primero realiza la unión sobre todas las variables y después elimina todas las variables ocultas.
- Se debería tener en cuenta el caso especial en el que el factor que se tiene que unir tiene una variable no condicionada (el docstring especifica qué hacer con mayor detalle).

Pregunta 7 (1 puntos): inferencia marginal

Dentro de `bayesAgents.py`, se debe usar la función `inference.inferenceByVariableElimination` que se ha completado en la pregunta anterior para completar la función `getMostLikelyFoodHousePosition`. Esta función debería computar la distribución marginal sobre las posiciones de la casa de comida y después devolver la posición con mayor probabilidad. El valor devuelto debería ser un diccionario que contenga un solo par llave-valor, `{FOOD_HOUSE_VAR: best_house_val}`, donde `best_house_val` es la posición más probable de `HOUSE_VALS`. Esto es utilizado por el Pacman Bayesiano, que deambula alrededor de forma aleatoria recolectando información por un número concreto de pasos de tiempo y después se dirige a la casa que tenga mayor probabilidad de contener comida.

Para comprobar el código:

```
python3 autograder.py -q q7
```

Pista: Se podría encontrar `Factor.getProbability(...)` y `Factor.getAllPossibleAssigmentDicts(...)` útiles.

Pregunta 8 (3 puntos): valor de información perfecta

El Pacman Bayesiano pasa una gran cantidad de tiempo deambulando de forma aleatoria, incluso cuando una mayor exploración no devuelve un valor adicional. ¿Cómo se podría hacer de forma más óptima?

Se evaluará Pacman VPI (valor de información perfecta) en una configuración más restrictiva: todo ha sido observado en el mundo menos los colores de una de las paredes de las casas. El Pacman VPI tiene 3 opciones:

1. Entrar directamente en la casa ya explorada.
2. Entrar directamente en la casa no explorada.
3. Explorar la parte externa de la casa occulta y después tomar una decisión sobre dónde ir.

Parte A

Primero se puede mirar la función `computeEnterValues`. Esta función calcula el valor esperado de entrar en las casas superiores derecha e izquierda. Otra vez, se puede realizar el cálculo de la función de inferencia que se ha escrito anteriormente en la red Bayesiana `self.BayesNet` para realizar todo el trabajo pesado ahí. Primero se computa *p(foodHouse = topLeft and ghostHouse = topRight | evidence)* y *p(foodHouse = topRight and ghostHouse = topLeft | evidence)*. Después se usan estas dos probabilidades para calcular las recompensas esperadas de entrar en las casas superiores derecha o izquierda. Se usa `WON_GAME_REWARD` y

`GHOST_COLLISION_REWARD` como la recompensa de entrar en foodHouse y ghostHouse respectivamente.

Parte B

Ahora se puede mirar la función `computeExploreValue`. Esta función calcula el valor que se espera de explorar todas las celdas ocultas y después tomar una decisión. En el código hay una función auxiliar, `getExplorationProbsAndOutcomes`, que devuelve una lista de las observaciones futuras que Pacman podría tomar y las probabilidades de cada una. Para calcular el valor de la información extra que ganará Pacman, se puede utilizar la fórmula siguiente:

$$E[value\ of\ exploration] = \Sigma p(evidence) \quad \max_{actions} E[value\ of\ action|evidence]$$

Se debe tener en consideración que $E[value\ of\ action|evidence]$ es exactamente la cantidad calculada por `computeEnterVals`, así que para calcular el valor de exploración se puede llamar a `computeEnterValues` de nuevo con la prueba hipotética devuelta por `getExplorationProbsAndOutcomes`.

Para comprobar el código:

```
python3 autograder.py -q q8
```

Pista: después de explorar, Pacman deberá calcular de nuevo el valor esperado de entrar en las casas derecha e izquierda. Dentro del código hay una función que hace justamente esto. La solución a `computeExploreValue` puede confiar en la solución de `computeEnterValues` para determinar el valor de observaciones futuras.

Entregables

- Archivos de código Python `factorOperations.py`, `inference.py` y `bayesAgents.py`.
- El código debe ir debidamente comentado explicando su funcionalidad de forma obligatoria. Debe ser legible y estar debidamente tabulado.
- Se utilizarán sistemas anticopia y se podrá requerir explicación individual de la práctica en caso de duda.
- Esta entrega se realizará en formato zip (incluyendo los archivos .py) vía Aula Virtual.
- Fecha de entrega: la entrega de la práctica se realizará **el martes 20 de diciembre** al finalizar la clase de prácticas (11:00-13:00). Durante la sesión, se propondrá una pequeña modificación a la práctica que deberá ser también entregada por los alumnos, por lo que la asistencia ese día será obligatoria.