INFO-F311 - Intelligence Artificielle
Projet 3: inférence bayesienne

Axel Abels - axel.abels@ulb.be
Yannick Mollinghen - Yannick.Molinghen@ulb.be

Année académique 2022–2023

Résumé

Pour ce troisième projet, vous aller implémenter les algorithmes permettant l'inférence bayesienne. Afin de concentrer votre effort sur l'implémentation des algorithmes, un squelette de code à compléter vous sera fourni.

1 Introduction

Le jeu Pacman consiste à guider Pac-Man à travers un labyrinthe afin de collecter de la nourriture. Pour ce troisième projet nous introduirons une observabilité partielle de l'environnement. Le pacman devra donc explorer son environnement afin d'acquérir plus de certitude quant aux actions à prendre.

2 Squelette Pac-Man

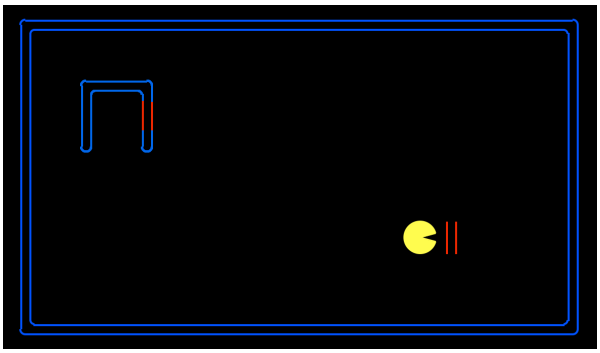
Pour ce troisième projet [1], vous trouverez sur l'Université Virtuelle un squelette de code qui vous permettra de coder les algorithmes demandés et d'en évaluer le fonctionnement. Cette base de code contient plusieurs fichiers qui assurent la simulation de l'environnement. Bien qu'il y ait de nombreux fichiers, vous ne devez travailler que sur `factorOperations.py`, `inference.py`, et `bayesAgents.py`. Ne modifiez pas d'autre fichiers que ceux-là car c'est les seuls fichiers que vous devrez soumettre.

3 La Chasse à la Nourriture

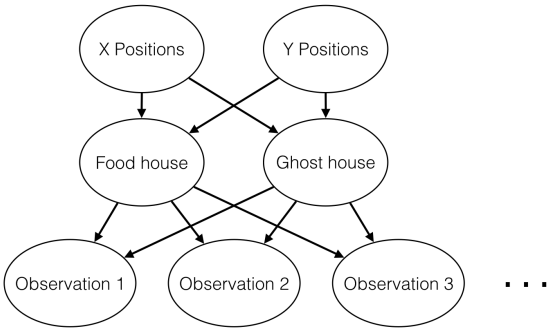
Pacman n'est plus doté d'une vision infaillible, il doit maintenant se contenter d'une vision portant uniquement sur les cases l'avoisinant. Il devra donc explorer afin d'acquérir plus d'information. La carte (illustrée par un example dans la figure 1a) contient deux maisons qui peuvent se trouver dans deux des quatre coins de la carte. Une de ces maisons contient la nourriture et sera probablement majoritairement bleue. L'autre contient un fantôme à éviter, et est probablement majoritairement rouge. En explorant les cotés de ces maisons, le pacman observera leurs couleurs et obtiendra plus de certitude quant à l'emplacement de la nourriture. En effet, s'il voit qu'une maison est majoritairement bleue, et que l'autre est majoritairement rouge, il saura que la nourriture se trouve probablement dans la première de ces maisons. L'exploration est cependant coûteuse, et il sera judicieux de choisir une des maisons dès qu'on aura acquis suffisamment d'évidence. A travers ce projet vous implémenterez les méthodes permettant d'accomplir ce raisonnement.

En particulier, nous vous demanderons d'initialiser un réseau bayesien, et ensuite d'implémenter des opérations dessus, tel que le join, l'élimination, et l'inférence.

1. Adapté du projet 4 du cours CS188 de l'Université de Berkeley (https://inst.eecs.berkeley.edu/~cs188/sp21/project4)



(A) Pacman explorant la carte. Pacman a déjà fait le tour de la maison à gauche en haut de la carte, et sait donc qu'elle est majoritairement bleue, elle contiendra donc probablement la nourriture. Il fait maintenant face à un des murs de la maison à droite en bas dont il connaît à présent la couleur. La couleur des autres murs ne sera observée que quand pacman passera à coté. Dans cet exemple la variable **X positions** prend probablement la valeur food-left, et la variable **Y positions** prend la valeur left-top.



(B) Le réseau bayesien à initialiser.

FIGURE 1 – Pacman et son réseau bayesien

3.1 Initialisation d'un Réseau Bayesien

Le réseau bayesien à initialiser est présenté à la figure 1b. Ce réseau reprend les variables suivantes :

- **X positions** détermine la position de la nourriture et peut prendre deux valeurs : *food-left* (la nourriture se trouve dans la maison de gauche) ou *ghost-left* (le fantome se trouve dans la maison de gauche, et la nourriture donc dans la maison de droite).
- **Y positions** détermine la position verticale des maisons et peut donc prendre quatre valeurs : *both-top* (les deux maisons sont en haut de la carte), *both-bottom* (les deux maisons sont en base de la carte), *left-top* (la maison gauche est en haut, et la maison droite est en bas, ou *left-bottom* (la maison gauche est en bas, et la maison droite est en haut).
- **Food house** détermine la position de la maison nourriture et est donc une fonction déterministe de **X positions** et de **Y positions**.
- **Ghost house** détermine la position de la maison fantome et est donc également une fonction déterministe de **X positions** et de **Y positions**.
- Les variables **Observation i** sont les observations que pacman peut faire. Il y a une telle observation par position où un mur pourrait se trouver. Chaque observation prend soit la valeur *none* s'il n'y a pas de mur, soit *bleu* ou *rouge* en fonction d'une distribution associée à la maison en question.

Dans `constructBayesNet` du fichier `bayesAgents.py` vous devrez initialiser ce réseau bayesien. Plus précisement, vous devrez donc

1. ajouter toutes les variables observation à la liste `obsVar` en suivant le template fourni dans le code.
2. ajouter tous les arcs du réseau à la liste `edges` sous forme de tuple. Par exemple, l'arc allant de la variable X positions (`X_POS_VAR`) à la variable Food house (`FOOD_HOUSE_VAR`) sera un tuple `(X_POS_VAR,FOOD_HOUSE_VAR)` à ajouter à cette liste. Vous retrouvez la définition des variables au début du fichier `bayesAgents.py`.
3. peupler les domaines des variables en ajoutant toutes les paires clé-valeur `(variable, valeurs possibles)` au dictionnaire `variableDomainsDict`. En plus des valeurs des observations définies par le template donné dans le code, vous retrouvez la définition des valeurs possible des autres variables au début du fichier `bayesAgents.py`.

Comme toujours, vous pouvez vérifier le fonctionnement de votre code à l'aide de l'`autograder` comme indiqué ci-dessous :

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

Notez qu'il n'y a pas de composante graphique pour ce projet.

Note: *Veillez bien à confiner votre code à l'interieur des bornes* `"*** YOUR CODE HERE ***"` `"*** END YOUR CODE HERE ***"`

3.2 Table de Probabilités Conditionnelles de Y positions

Complétez la fonction `fillYCPT` du fichier `bayesAgents.py` afin de peupler la table de probabilités conditionnelles de la variable Y positions. Pour ce faire, vous devrez utiliser la méthode `setProbability` de la classe `Factor` sur l'object `yFactor` afin d'attribuer la probabilité correspondante à chaque dictionnaire d'assignation possible. L'assignation répond donc implicitement à la question "quelle est la probabilité que les variables en clef du dictionnaire prennent les valeurs correspondantes ?". Vous retrouverez les definitions des probabilités à utiliser au sommet du fichier `bayesAgents.py`.

Testez votre code avec l'`autograder` comme indiqué ci-dessous :

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

3.3 Joindre des Facteurs

Afin de faire de l'inférence, vous devrez joindre des facteurs et éliminer des variables. Nous vous demandons d'abord d'implémenter le join dans `joinFactors` du fichier `factorOperations.py`. Cette méthode prend en argument une liste de facteurs et devra retourner un nouveau facteur : le résultat du join. Notez que le join permet l'application de la règle du produit. Par exemple, $P(X|Y)P(Y) = P(X,Y)$, et on aura donc que $joinFactors([P(X|Y), P(Y)]) = P(X,Y)$.

D'autres exemples :

— $joinFactors([P(V,W|X,Y,Z), P(X,Y|Z)]) = P(V,W,X,Y|Z)$

— $joinFactors([P(X|Y,Z), P(Y)]) = P(X,Y|Z)$

— $joinFactors([P(V|W), P(X|Y), P(Z)]) = P(V,X,Z|W,Y)$

En vous basant sur ces exemples, quelles variables sont conditionnées (et se retrouveront donc dans `inputConditionedVariables`) ? Quelles variables sont non-conditionnées (qui se retrouveront donc dans `inputUnconditionedVariables`) ?

Vous devrez assigner les probabilities à l'aide de la méthode `setProbability`. Vous retrouverez des exemples de calcul de probabilités à la suite d'un join dans vos notes de cours.

Note: *Le constructeur Factor prend un dictionnaire de domaines en argument. Celui-ci étant partagé pour tous les factors, vous pouvez simplement passer le dictionnaire de domaines d'un des factors fournis en argument à la fonction* `joinFactors`.

Note: *La somme des probabilités ne sera pas nécessairement* 1.

Vous pouvez tester votre implémentation avec l'`autograder` avec la commande

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

3.4 Eliminer des facteurs

Complétez la fonction `eliminate` du fichier `factorOperations.py` afin de permettre l'élimination (marginalisation) d'une variable d'un facteur. Pour rappel, afin d'éliminer une variable, il suffit de sommer les probabilités sur l'ensemble des valeurs que la variable peut prendre. Quelques facteurs attendus après un appel à la fonction `eliminate` :

— $eliminate(P(X, Y|Z), Y) = P(X|Z)$
— $eliminate(P(X, Y|Z), X) = P(Y|Z)$

Vous devrez assigner les probabilities à l'aide de la méthode `setProbability`. Vous retrouverez des exemples de calcul de probabilités à la suite d'une marginalisation dans vos notes de cours.

Vous pouvez tester votre fonction avec la commande

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

3.5 Normaliser

Afin de pouvoir normaliser les probabilités (tel que leur somme vaille 1), implémentez la fonction `normalize` du fichier `factorOperations.py`. Cette fonction prend un facteur en argument et renvoi une copie normalisée. Notez que si la somme des probabilités vaut 0 vous devrez simplement renvoyer `None`.

> **Note:** *Veuillez vous référer au docstring de la fonction afin d'établir les variables conditionnées attendues.*

Vous pouvez tester votre fonction avec la commande

```
$ python autograder.py -q q5
```

3.6 Inférence par élimination

En vous inspirant de `inferenceByEnumeration`, complétez la fonction `inferenceByVariableElimination` du fichier `inference.py` afin de permettre l'inférence par élimination. En partant de la liste complète de facteurs `currentFactorsList`, vous devrez alterner les join et eliminate sur les variables dont l'ordre est indiqué par `eliminationOrder`.

> **Note:** *Si l'unique variable inconditionnée d'un facteur est la variable à éliminer, il faudra simplement passer ce facteur.*

> **Note:** *Les probabilités du facteur retourné devront sommer à* 1.

Vous pouvez tester votre fonction avec la commande

```
$ python autograder.py -q q6
```

3.7 Trouver la Position de la Nourriture

Complétez la fonction `getMostLikelyFoodHousePosition` du fichier `bayesAgents.py` afin de renvoyer la position la plus probable de la nourriture. Pour ce faire, vous devrez faire appel à la fonction `inference.inferenceByVariableElimination` que vous avez implémentée au point précédent. Le but est donc de calculer les probabilités marginales de la variable Food house et d'ensuite renvoyer la valeur dont la probabilité est maximale.

> **Note:** *Votre fonction doit retourner uniquement la valeur (non pas un dictionnaire d'attribution).*

Vous pouvez tester votre fonction avec la commande

```
$ python autograder.py -q q7
```

3.8 La Valeur de l'Information Parfaite

Afin d'estimer le besoin d'exploration, vous implémenterez pour cette dernière question deux fonctions qui permettront d'évaluer les gains attendus de différentes actions. Pour cette question nous partirons d'une situation où le pacman a déjà exploré une maison, et connaît donc ses couleurs. Il a alors trois choix : (i) entrer dans la maison explorée, (ii) entrer dans la maison non-explorée, ou (iii) explorer d'avantage afin d'obtenir plus de certitude quant à la position de la nourriture. Afin d'identifier la meilleure action, nous devrons estimer si l'exploration vaut le coup. Il s'agira donc d'estimer si l'on a déjà suffisamment d'évidence, ou s'il vaut mieux payer le coût d'exploration afin de s'assurer qu'on choisi correctement la maison nourriture. Pour ce faire, vous devrez implémenter deux fonctions, `computeEnterValues` et `computeExploreValue` du fichier `bayesAgents.py`.

La première de ces fonctions, `computeEnterValues` calculera la valeur attendue d'entrer immédiatement dans une des deux maisons. Elle renverra donc un tuple `(leftExpectedValue, rightExpectedValue)`. Pour ce faire, vous pouvez calculer les probabilités associées à la présence de nourriture/fantôme dans chaque maison en utilisant les méthodes précédemment implémentées. En combinaison avec les variables `WON_GAME_REWARD` et `GHOST_COLLISION_REWARD` vous pourrez alors calculer les valeurs attendues.

Ensuite, calculez le coût d'une exploration complète suivie d'un choix de maison dans `computeExploreValue`. Pour ce faire, vous devrez faire appel à la méthode `getExplorationProbsAndOutcomes`. Cette méthode renvoie une liste de tuples `(probabilité, observation)` qui reprend les futures observations que pacman peut faire ainsi que leurs probabilités. Vous pouvez alors utiliser ces observations afin de calculer la valeur attendue d'une exploration suivie d'une action. Plus précisément, vous devriez utiliser la formule suivante :

$$E[valeur\ d'exploration] = \sum_{observations} p(observation) \max_{actions} E[valeur\ de\ l'action|observation]$$

C'est cette valeur que la fonction devra retourner.

> **Note:** *Les actions sur lesquelles vous maximisez sont l'entrée dans la maison gauche ou droite. Leur valeur peut donc être calculée à l'aide de `computeEnterValues` que vous venez d'implémenter.*

Vous pouvez tester votre fonction avec la commande

```
$ python autograder.py -q q8
```

4 Évaluation

Vous pouvez tester l'entièreté de votre code à l'aide de

```
$ python autograder.py
```

Cette note de fonctionnement contribuera à la note finale pour ce projet. En plus du fonctionnement nous vous demandons également de soigner votre code, en appliquant les conventions de programmation vues tout au long de votre parcours académique. Pensez donc à éviter la duplication de code, et à commenter votre code pertinemment.

Veillez à la bonne appellation de variables et de fonctions (si vous en ajoutez), non seulement en étant cohérent (en suivant la convention du squelette : `camelCase` pour les variables et méthodes/fonctions, et le `PascalCase` pour les noms de classes), mais également en choisissant des noms qui aident à la compréhension du code.

5 Délivrables

Il vous est demandé de ne soumettre que les fichiers `factorOperations.py`, `inference.py`, et `bayesAgents.py`. Nous ne prendrons donc pas en compte tout autre fichier modifié.

Le code remis doit être le résultat d'un travail individuel. Tout emprunt non mentionné est un plagiat et sera pénalisé.

https://bib.ulb.be/fr/support/boite-a-outils/evitez-le-plagiat

Consignes pour la remise du projet

À respecter scrupuleusement !

1. Le projet est à remettre sur l'UV uniquement.
2. Les fichiers `factorOperations.py`, `inference.py`, et `bayesAgents.py` complétés doivent être remis.
3. Votre code doit être **commenté** lorsque c'est pertinent et compatible avec Python 3.8.
4. Vous devez respecter les modalités suivantes :
   - Poster vos fichiers sources sur l'UV.
   - Date : avant le **dimanche 20 novembre à 23h59**
5. Posez de préférence vos questions après le cours du lundi où les assistants en charge du projet seront disponibles pour y répondre.