Université de Caen Normandie
Département d'informatique
L3 informatique, 2018 – 2019
Unité MIM5E2

Aide à la décision et intelligence artificielle

Travaux pratiques

Grégory Bonnet

---

1 Implémentation d'un problème de planification

En vous fondant sur le travail d'implémentation d'un problème de satisfaction de contrainte, implémentez un problème de planification dans un paquetage `planning`. Vous devez développer :

- Une classe `State` qui représente une affectation d'un ensemble de `Variable`.
- Une classe `Action` qui représente (comme son nom l'indique) une action. Cette classe doit permettre d'exprimer des affectations de `Variable` représentant les préconditions et les effets. Pensez à utiliser ou vous inspirer de la classe `Rule`.
- Une classe `PlanningProblem` qui représente un problème de planification. Cette classe doit permettre d'exprimer l'état initial, les états finaux et la liste des actions à disposition du planificateur.

Implémentez les fonctions (au bon niveau dans la hiérarchie des classes) :

- `is_applicable` qui vérifie si une action est applicable dans un état donné.
- `satisfies` qui vérifie si un état donné satisfait les états finaux.
- `apply` qui applique une action donnée dans un état donné.

Les deux premières fonctions retournent un booléen. Pour la dernière fonction, vous prendez garde à bien retourner une nouvelle instance de l'état.

2 Application : une chaîne de montage de voiture

Dans le paquetage `example`, implémentez un classe `HealthCare` qui simule la prise en charge d'un patient. Le problème par :

1. des variables booléennes représentant les maladies : `ANGINA`, `FLU`, `POX`, `PLAGUE`

2. des variables à niveau dans $\{\text{high}, \text{medium}, \text{low}, \text{none}\}$ représentant des symptômes `FEVER`, `COUGH`, `BUTTONS`,

3. des actions qui représentent trois sirops qui réduisent chacun un symptôme distinct d'un niveau. Par exemple,

$$\texttt{SYRUP\_BUTTONS\_HIGH} : \texttt{BUTTONS} = \text{high} \implies \texttt{BUTTONS} = \text{medium}$$

4. des actions qui représentant n médicaments expérimentaux générés aléatoirement qui traitent les symptÃ´mes. Ce nombre de médicament n est un paramètre de votre classe. Appelez ces médicaments `MEDICINE_A`, `MEDICINE_B`, ..., `MEDICINE_N`. Chaque médicament permet de réduire un symptôme donné à none mais fixe chacun des autres à un niveau aléatoire. Par exemple, nous pourrions considérer les médicaments suivants.

$$\texttt{MEDICINE\_A} : \top \implies \texttt{FEVER} = \text{none}$$
$$\texttt{MEDICINE\_A} : \top \implies \texttt{COUGH} = \text{low}$$
$$\texttt{MEDICINE\_A} : \top \implies \texttt{BUTTON} = \text{low}$$

5. une action de guérison :

$$\texttt{HEAL} : \texttt{FEVER} = \text{none} \wedge \texttt{COUGH} = \text{none} \wedge \texttt{BUTTONS} = \text{none} \implies \texttt{ANGINA} = \bot$$
$$\texttt{HEAL} : \texttt{FEVER} = \text{none} \wedge \texttt{COUGH} = \text{none} \wedge \texttt{BUTTONS} = \text{none} \implies \texttt{FLU} = \bot$$
$$\texttt{HEAL} : \texttt{FEVER} = \text{none} \wedge \texttt{COUGH} = \text{none} \wedge \texttt{BUTTONS} = \text{none} \implies \texttt{POX} = \bot$$
$$\texttt{HEAL} : \texttt{FEVER} = \text{none} \wedge \texttt{COUGH} = \text{none} \wedge \texttt{BUTTONS} = \text{none} \implies \texttt{PLAGUE} = \bot$$

Au besoin, si vous désirez complexifier le problème, vous pouvez rendre certains medicaments inutilisable pour certaines maladies. Pour cela, il vous suffit de changer les préconditions des actions associées.

3 Instantiation d'un problème de planification

Implémentez dans `HealtCare` une fonction qui génère aléatoirement un état intial : une maladie et un ensemble de symptomes avec un certain niveau. L'état but est toujours le même : la maladie à $\bot$ et tous les symptômes à none.

4 Implémentation des algorithmes de résolution

Implémentez dans la classe `PlanningProblem` les trois algorithmes suivants :

1. recherche en profondeur,

2. recherche en largeur.

Si vous avez le temps, vous pouvez aussi implémenter un algorithme de recherche en profondeur itérative.

Afin d'expérimenter ces algorithmes, implémentez des sondes qui comptent le nombre de nœuds explorés par chaque algorithme. Générez un grand nombre de problèmes et calculez, pour chaque algorithme, le nombre moyen de nœuds explorés avant d'atteindre un but. Indiquez les résultats dans un commentaire d'en-tête de votre classe `PlanningProblem`.

5 Implémentation du coût des actions

Étendez la classe `PlanningProblem` en une classe `PlanningProblemWithCost` qui intègre une fonction de coût pour chaque action. Donnez un coût à toutes les actions :

— les actions de prise de sirop coûtent 2,

— les actions de prise de médicament expérimentaux coûtent 1.

Vous pouvez modifier les coûts à des fins expérimentales.

6 Implémentation des algorithmes de recherche

Implémentez les deux algorithmes suivants :

1. algorithme de Dijkstra,
2. algorithme A*.

Pour l'algorithme A*, implémentez un patron de conception de type stratégie appelé `Heuristic`. Cette classe doit contenir les méthodes permettant de calculer l'heuristique associée à un état. Définissez deux heuristiques :

1. `SimpleHeuristic` qui fait la somme des niveaux des symptômes,
2. `InformedHeuristic` qui doit être plus informée que `SimpleHeuristic`.

Réutilisez les sondes développpées précédemment pour compter le nombre moyen de nœuds explorés par chaque algorithme et chaque heuristique. Comme précédemment, indiquez les résultats dans un commentaire d'en-tête de votre classe `PlanningProblemWithCost`.