Projet MADI

Clémence BOURGUE et Ariana CARNIELLI

16 janvier 2021

Created on Mon Dec 7 21 :01 :59 2020

Projet MADI Algorithmes pour la planification dans le risque

@author : Clémence BOURGUE

@author : Ariana CARNIELLI

class projet_madi.**Grille**(*nb_lig*, *nb_col*, *tab_cost=[1, 2, 3, 4]*, *p=1*, *proba_coul=None*, *proba_mur=0*, *proba_nb=None*)

Représente la grille. La grille est représentée par 2 arrays numpy 2D d'entiers, une où chaque entier représente une couleur et une ou chaque entier représente un poids entre 1 et 9. Initialise la grille des couleurs et des poids de façon aléatoire en utilisant la liste de probabilités proba_coul pour la grille de couleurs et la liste de probabilités proba_nb pour les poids.

Paramètres

- **nb_lig** (*int*) – Nombre de lignes de la grille.
- **nb_col** (*int*) – Nombre de colonnes de la grille.
- **tab_cost** (*list (int)*) – Liste des coûts de chaque couleur. Le défaut est [1, 2, 3, 4].
- **p** (*float*) – Probabilité d'atteindre la case cible. Le défaut vaut 1.
- **proba_coul** (*list (float)*) – Liste des probabilités pour l'occurrence de chaque couleur (taille égale à la taille de tab_cost) si None, probabilité uniforme. Le défaut est None.
- **proba_mur** (*float*) – Probabilité d'occurrence des murs. Le défaut vaut 0.
- **proba_nb** (*list (float)*) –
- **des probabilités pour l'occurrence de chaque chiffre entre 1** (*Liste*) –
- **9. Si None** (*et*) –
- **uniforme. Le défaut est None.** (*probabilité*) –

tab

Tableau représentant la grille. Chaque case contient un nombre indiquant à quelle couleur la case est associée. Les cases avec -1 contiennent des murs.

Type numpy.ndarray

tab_cost

Liste des coûts de chaque couleur.

Type list(int)

p

Probabilité d'atteindre la case cible.

Type float

chiffre

Tableau représentant les prix de la grille. Chaque case contient un chiffre entre 1 et 9.

Type numpy.ndarray

case_cout(*i*, *j*, *mode*)

Retourne le coût de la case (i, j) d'après le mode de calcul. Deux modes : 'couleur' et 'somme_chiffre'. Mode 'couleur' retourne le coût basé sur le tableau tab_coul. Mode 'somme_chiffre' retourne le coût basé sur le tableau chiffre. Mode "chiffre" retourne un tuple avec le coût basé sur le tableau chiffre et la couleur de la case.

Paramètres

- **i** (*int*) – Indice de ligne de la case.
- **j** (*int*) – Indice de colonne de la case.
- **mode** (*String*) – Le mode de calcul du coût. "couleur", "somme_chiffre" ou "chiffre".

Renvoie Coût de la case d'après le mode choisi.

Type renvoyé int ou tuple(int, int)

case_possible(*i*, *j*)

Retourne True si la case (i, j) est atteignable, False sinon.

Paramètres
- **i** (*int*) – Indice de ligne de la case.
- **j** (*int*) – Indice de colonne de la case.

Renvoie True si case atteignable, False sinon.

Type renvoyé boolean

est_possible()
Teste si la case but de la grille est atteignable depuis la case initiale (plus en haut et à gauche).

Renvoie _ – True si la case but est atteignable, False sinon.

Type renvoyé bool

proba_trans(*i*, *j*, *action*)
Calcule les cases atteignables a partir d'une case (i, j) quelconque en prenant l'action a, et les respectives probabilités.

Paramètres
- **i** (*int*) – Indice de ligne de la case.
- **j** (*int*) – Indice de colonne de la case.
- **a** (*int*) – L'action à réaliser. L'encodage est 0-haut, 1-droite, 2-bas, 3-gauche.

Renvoie **cases_p** – Dictionnaire indexé par les cases atteignables et contenant les probabilités d'atteindre chaque case.

Type renvoyé defaultdict((int, int), float)

proba_trans_arr(*i*, *j*)
Calcule un dictionnaire avec les probabilités T((i', j'), a, (i, j)), utilisé dans les calculs des PL. Les clés sont des triplets (i', j', a) des cases qui peuvent atteindre une case (i, j) en faisant l'action a et les valeurs sont les probabilités d'atteindre la case (i, j).

Paramètres
- **i** (*int*) – Indice de ligne de la case.
- **j** (*int*) – Indice de colonne de la case.

Renvoie **cases_p** – Dictionnaire indexé par les cases et actions et contenant les probabilités d'atteindre la case (i, j).

Type renvoyé defaultdict((int, int), float)

class projet_madi.**Visualisation**(*grille*, *tab_coul*)
Classe créée pour faciliter la visualisation des grilles et la movimentation du robot. Permet la visualisation des deux types de grille avec un guidage manuel du robot par les flèches du clavier, en montrant les coûts accrus en temps réel. On peut aussi visualiser des flèches indiquant l'action optimale d'après une politique stationnaire déterministe et utiliser une politique déterministe ou mixte pour le guidage automatique du robot en tapant la touche 'espace'.

Paramètres
- **grille** (Grille) – La grille à visualiser.
- **tab_coul** (*list(string)*) – List avec les codes hexadécimaux des couleurs utilisés dans la visualisation de la grille. Doit avoir au moins la même quantité que des couleurs dans la grille.

grille
La grille à visualiser.

Type *Grille*

tab_coul
List avec les codes hexadécimaux des couleurs utilisés dans la visualisation de la grille. Doit avoir au moins la même quantité que des couleurs dans la grille.

Type list(string)

case_px
La taille en pixels d'une case de la grille. Le défaut est 40.

Type int

strategy
Tableau représentant une stratégie. Chaque case contient soit un nombre entre 0 et 3 indiquant à quelle case voisine le robot doit aller ou une distribution de probabilités pour les cases voisines.

Type numpy.ndarray

_clavier(*event*)
Controle du robot par les flèches du clavier et action automatique d'après une stratégie avec la touche espace.

_dessin_chiffre()
Dessine les cases de la grille dans le mode "chiffre". On a un chiffre coloré dans chaque case.

_dessin_couleur()
Dessine les cases de la grille dans le mode "couleur". On a un petit circle coloré dans chaque case.

_dessin_fleche()
Dessine les flèches indiquant la meilleure choix si on a une stratégie pure. On dessine aussi une flèche en dessus du pion.

_dessin_grid(*largeur*, *hauteur*)
Dessine les droites du grid de la grille.

_move(*action*)
Bouge le robot et la flèche en dessus si elle existe.

_reinitialize()
Réinitialise la visualisation de la grille.

view(*case_px=40*, *mode='couleur'*, *strategy=None*)
Permet la visualisation de la grille dans une fenêtre à l'aide du module Tkinter.

Paramètres
- **case_px** (*int*) – La taille en pixels d'une case de la grille. Le défaut est 40.
- **mode** (*string*) – Le mode de visualisation. Peut être "couleur" ou "chiffre". The default is "couleur". Le mode "couleur" correspond à la grille présenté dans la partie 2 et 3. Le mode "chiffre" correspond à la grille présenté dans la partie 4.
- **strategy** (*numpy.ndarray*) – Tableau représentant une stratégie. Peut être 2D (si stratégie pure) ou 3D (si stratégie mixte). The default is None.

projet_madi.**pol_pl_mixte**(*grille*, *gamma*, *M*, *mode='couleur'*, *verbose=False*)
Calcule la stratégie optimale pour une grille donnée, avec un gamma et une récompense finale passées en argument, en utilisant un PL. Le mode indique quels coûts utiliser pour le calcul. La stratégie calculé est "mixte" (une distribution de probabilités).

Paramètres
- **grille** (*Grille*) – La Grille pour laquelle on calcule la stratégie optimale.
- **gamma** (*float*) – Le gamma (taux d'amortissement) utilisé dans le calcul.
- **M** (*int*) – La récompense de la case but.
- **mode** (*String*) – Avec quel coût calculer la stratégie. Deux modes : "couleur" et "somme_chiffre". "couleur" calcule la stratégie avec les coûts en tab_cost (partie 2 de l'énoncé). "somme_chiffre" calcule la stratégie avec la somme des coûts en chiffre (partie 4a de l'enoncé). Le défaut est "couleur".
- **verbose** (*bool*) – If True, shows the resolution of the PL calculated by Gurobi. Le défaut est False.

Renvoie
- **pol** (*numpy.ndarray*) – Tableau 3D représentant une stratégie mixte.
- **obj_val** (*float ou list(float)*) – Valeur de la fonction objectif à l'optimum dans le cas du mode "couleur", ou liste avec les valeurs selon chaque critère dans le mode "somme_chiffre".

projet_madi.**pol_pl_mixte_mo**(*grille*, *gamma*, *M*, *verbose=False*)
Calcule la stratégie optimale pour une grille donnée, avec un gamma et une récompense finale passées en argument, en utilisant un PL. Cette fonction est spécifique pour la grille multi-objectifs avec comme fonction

d'agrégation un critère max min(celle de la partie 4b). La stratégie calculé est "mixte" (une distribution de probabilités).

Paramètres

- **grille** (Grille) – La Grille pour laquelle on calcule la stratégie optimale.
- **gamma** (*float*) – Le gamma (taux d'amortissement) utilisé dans le calcul.
- **M** (*int*) – La récompense de la case but.
- **verbose** (*bool*) – If True, shows the resolution of the PL calculated by Gurobi. Le défaut est False.

Renvoie

- **pol** (*numpy.ndarray*) – Tableau 3D représentant une stratégie mixte.
- **obj_val** (*list(float)*) – Valeur à l'optimum de l'objectif selon chaque critère.

projet_madi.**pol_pl_pure**(*grille*, *gamma*, *M*, *mode='couleur'*, *verbose=False*)

Calcule la stratégie optimale pour une grille donnée, avec un gamma et une récompense finale passées en argument, en utilisant un PLNE. Le mode indique quels coûts utiliser pour le calcul. La stratégie calculé est "pure" (une seule valeur par case).

Paramètres

- **grille** (Grille) – La Grille pour laquelle on calcule la stratégie optimale.
- **gamma** (*float*) – Le gamma (taux d'amortissement) utilisé dans le calcul.
- **M** (*int*) – La récompense de la case but.
- **mode** (*String*) – Avec quel coût calculer la stratégie. Deux modes : "couleur" et "somme_chiffre". "couleur" calcule la stratégie avec les coûts en tab_cost (partie 2 de l'énoncé). "somme_chiffre" calcule la stratégie avec la somme des coûts en chiffre (partie 4a de l'enoncé). Le défaut est "couleur".
- **verbose** (*bool*) – If True, shows the resolution of the PLNE calculated by Gurobi. Le défaut est False.

Renvoie

- **pol** (*numpy.ndarray*) – Tableau 2D représentant une stratégie pure.
- **obj_val** (*float*) – Valeur de la fonction objectif à l'optimum.

projet_madi.**pol_valeur**(*grille*, *gamma*, *M*, *eps=1e-05*, *mode='couleur'*)

Calcule la stratégie optimale pour une grille donnée avec un gamma et une récompense finale passées en argument, en utilisant l'algorithme d'itération de la valeur. Le critère d'arrêt de l'approximation est aussi passé en argument, ainsi que le mode (i.e quel coûts utiliser pour le calcul).

Paramètres

- **grille** (Grille) – La Grille pour laquelle on calcule la stratégie optimale.
- **gamma** (*float*) – Le gamma (taux d'amortissement) utilisé dans le calcul.
- **M** (*int*) – La récompense de la case but.
- **eps** (*float*) – Le critère d'arrêt utilisé dans le calcul. Le défaut est 1e-5.
- **mode** (*String*) – Avec quel coût calculer la stratégie. Deux modes : "couleur" et "somme_chiffre". "couleur" calcule la stratégie avec les coûts en tab_cost (partie 2 de l'énoncé). "somme_chiffre" calcule la stratégie avec la somme des coûts en chiffre (partie 4a de l'enoncé). Le défaut est "couleur".

Renvoie

- **pol** (*numpy.ndarray*) – Tableau 2D représentant une stratégie pure.
- **cpt** (*int*) – La quantité d'itérations avant la convergence de l'algorithme.

projet_madi.**simulation**(*grille*, *strategy*, *gamma*, *bonus*, *mode='couleur'*, *maxIter=10000*, *init_robot=0, 0*)

Simule une stratégie pure ou mixte sur une grille.

Paramètres

- **grille** (Grille) – La grille sur laquelle on veut tester la stratégie.
- **strategy** (*numpy.ndarray*) – La stratégie à tester.
- **gamma** (*float*) – Le gamma (taux d'amortissement) utilisé dans le calcul.

— **bonus** (*int*) – Le bonus reçu (une seule fois) dans la case cible. Lié avec le paramètre M par bonus = M/(1-gamma).
— **mode** (*string*) – Le mode de calcul : "couleur" pour calculer un cout scalaire avec le cout associé à chaque couleur, "chiffre" pour calculer un cout vectoriel selon les valeurs de grille.chiffre et les différents critères associés à chaque couleur.
— **maxIter** (*int*) – Nombre maximal d'itérations.
— **init_robot** (*tuple(int, int)*) – Position initiale du robot.

Renvoie Le cout observé (float en mode "couleur", liste de float en mode "chiffre").

Type renvoyé float ou list(float)

projet_madi.**tester_iterations**(*fonction*, *list_grille*, ***kwargs*)

Implémente le test de quantité d'itérations moyenne démandé à l'énoncé.

Paramètres
— **fonction** (*Function*) – La fonction qu'on veut tester le temps d'exécution. Il faut qu'elle prenne une grille en argument.
— **list_grille** (*list(Grille)*) – Liste avec toutes les Grilles qui doivent être testés.
— ****kwargs** – Keyword arguments necessaires à la fonction testée.

Renvoie Le nombre moyen d'itérations de la fonction passée en argument.

Type renvoyé float

projet_madi.**tester_temps**(*fonction*, *list_grille*, *repeat=10*, ***kwargs*)

Implémente le test de temps de calcul moyen démandé à l'énoncé.

Paramètres
— **fonction** (*Function*) – La fonction qu'on veut tester le temps d'exécution. Il faut qu'elle prenne une grille en argument.
— **list_grille** (*list(Grille)*) – Liste avec toutes les Grilles qui doivent être testés.
— **repeat** (*int*) – La quantité de fois chaque Grille aura une stratégie calculé. Le défaut est 10.
— ****kwargs** – Keyword arguments necessaires à la fonction testée.

Renvoie Le temps moyen de calcul de la fonction passée en argument.

Type renvoyé float