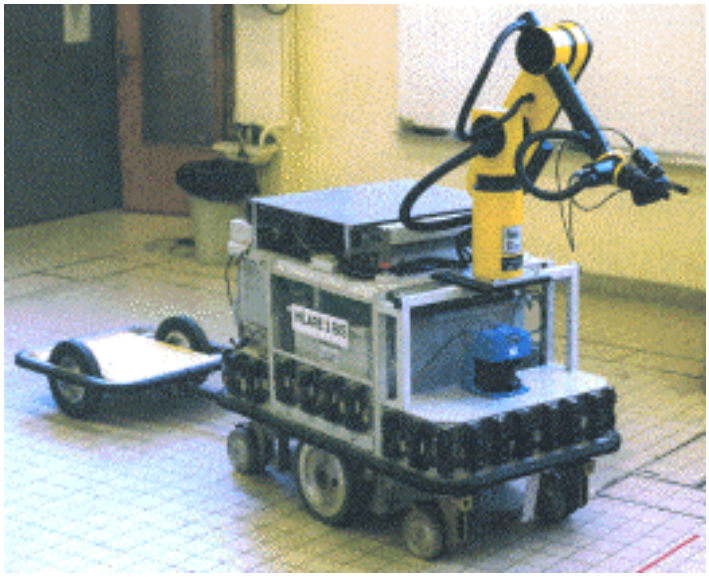
Apprentissage par Renforcement (Reinforcement learning) Approche : Programmation dynamique

T. AL-ANI
A²SI-ESIEE-PARIS

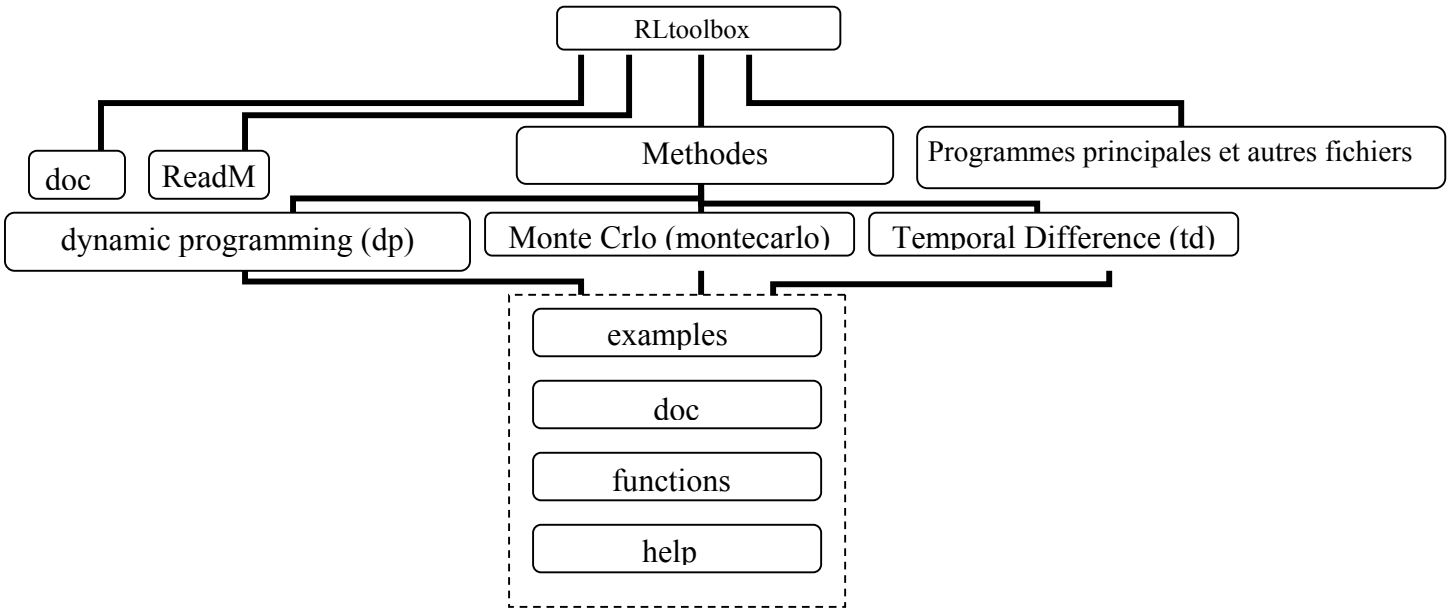


Sommaire

Chapitre 1

BOITE A OUTILS DE L'APPRENTISSAGE PAR RENFORCEMENT : (RLtoolbox)

Le toolbox Rltoolbox_V.1 est organisé comme suit :



ReadMe: installation et quelques guides générales

doc : contient quelques tutoriaux généraux sur l'apprentissage par renforcement.

Programmes principales et autres fichiers : RLtoolbox.sce, RLtoolbox.tcl, RLtoolbox.gif, RLtoolbox.pdf.

Chacun des répertoires approches est composé des éléments suivants:

Le répertoire **examples** contient toutes les fichiers de démonstrations.

Le répertoire **help** contient toutes les fichiers d'aide.

Le répertoire **doc** contient quelques rapports téchniques.

Le répertoire **functions** contient tous les programmes utilisés dans le cadre de l'Apprentissage par Renforcement. Pour l'approche (dp) les programmes sont les suivants :

Evaluation de la Politique itérative (Iterative_Policy_Evaluation)

L'algorithme a pour but de rechercher la fonction de valeur V^{π} pour une politique choisie arbitrairement π.
Pour produire chaque approximation successive V_{k+1} de V_k, l'algorithme applique la même opération à chaque état s : il remplace l'ancienne valeur de s par une nouvelle obtenue des anciennes valeurs des états successeurs à s, et de la récompense immédiate attendue, tout le long du des « one-step transitions » sous la politique en cours d'évaluation. C'est ce qu'on appelle le *full backup,* il est appelé ainsi car il est basé sur les tous états suivants possibles plutôt que sur un seul.
Chaque itération effectue un retour sur la valeur de chaque état une fois pour produire la nouvelle fonction de valeur V_{k+1}.
Pour implémenter l'algorithme de programmation dynamique on doit utiliser deux variables V_k et V_{k+1}. Un pour les anciennes valeurs l'autre pour les nouvelles

Amélioration de la politique itérative (Iterative_Policy_Improving) &
Amélioration des actions de la politique itérative (Iterative_Policy_Improving_Action_Value)

On dispose maintenant de la fonction de valeur d'état V^{π} déterminée pour une politique donnée π. Pour un état s on voudrait savoir s'il est possible de changer la politique pour choisir une action $a \neq \pi(s)$. Maintenant qu'on connaît la rentabilité de la politique courante pour s, serait il plus ou moins bénéfique si on opte pour une nouvelle politique ?
Un manière de répondre à la question est de considérer une sélection de a et reprendre la politique π ensuite. On note alors $Q^{\pi}(s,a)$ cette nouvelle fonction de valeur d'état. Le critère est la comparaison avec $V^{\pi}(s)$. Si c'est plus grand alors il serait mieux de sélectionner la nouvelle action a en s et reprendre π ensuite plutôt que de suivre π tout le temps. Choisir la nouvelle action a chaque fois que s est rencontré engendre une nouvelle politique qui sera meilleure que π.

Valeur de l'itération (Value_Iteration)

Un inconvénient de l'itération de la politique est que chaque itération comprend une évaluation de la politique. Si l'évaluation de la politique est faite itérativement alors la convergence vers V^{π} n'a lieu que vers la limite.
Il est inutile d'attendre jusqu'à la fin. En effet l'itération de la politique peut être tronquée par différents moyens sans perdre la garantie de la convergence de l'itération de la politique. Un cas important est quand l'évaluation de la politique est stoppée après un seul retour d'état (*backup*) sur chaque état. C'est l'algorithme de la Value iteration.

Vérification de paramètres entrée par l'utilisateur (verif_param2)

Le Robot a pour but de collecter un maximum de canettes sachant qu'il possède deux états (batterie chargée, batterie déchargée) et trois actions (collecter les canettes ; se charger ; attendre que quelqu'un pose une canette). Les programmes ont permis de montrer que la politique optimale est que le robot collecte les canettes lorsqu'il est chargé et se recharge à chaque fois qu'il est déchargé [1, 2].

Chapitre 2

Exemples

I) Les interfaces graphiques sous Scilab

Les échanges graphiques lors de l'utilisation de la Toolbox se passent par des fenêtres de messages annexes. Après le lancement du programme, il n'y a plus à se référer à la fenêtre d'exécution de Scilab. Une série de messages vous demande ce que vous voulez faire et vous aiguille jusqu'à la fin de l'exécution de la toolbox.
Ce système de messages était déjà en partie implémenté dans les programmes que nous avons récupérés mais nous avons travaillé dessus afin de les rendre plus clair et plus conviviaux. Nous avons également du ajouter des étapes dans les choix de procédure pour pouvoir intégrer plusieurs démonstrations illustrant le « reinforcement learning ».
Ensuite, au-delà de l'interface de la toolbox, il y a les interfaces des exemples eux-mêmes. Ici, il y a toujours des messages explicatifs tout au long de l'utilisation des exemples. Il s'agit surtout ici d'expliquer le fonctionnement et l'objectif ou les objectifs des exemples que nous avons intégrés au logiciel. Pour la présentation des résultats, nous avons également fait appel aux fonctions graphiques de Scilab. Cela nous a permis de présenter les choses beaucoup plus clairement et efficacement que par de simples messages (un bon schéma vaut mieux qu'un long discours). Nous avons donc utilisé une fenêtre graphique et il est important de le noter car celle-ci apparaît *en arrière plan* par rapport à la fenêtre d'exécution Scilab. De plus, elle est très souvent juste derrière cette fenêtre et se retrouve donc masquée. C'est un problème inhérent à Scilab qu'il nous a été impossible de résoudre.
La présentation actuelle de la toolbox impose à l'utilisateur de repasser à travers la toolbox pour lancer un deuxième exemple ou même pour relancer le même exemple deux fois de suite. On peut facilement changer ça dans le programme, ce changement ne nous a pas paru important mais selon l'utilisation de la toolbox, il pourrait être intéressant de pouvoir se déplacer entre les exemples sans avoir à revoir toute la procédure. Tout dépend de l'utilisation future de la toolbox.

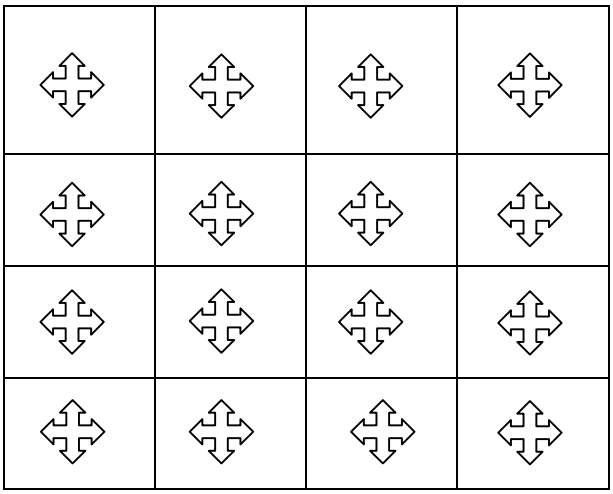
II) Premier exemple : The Grid

Considérons la grille 4*4 ci-dessous :

15	1	2	3
4	5	6	7
8	9	10	11

12	13	14	16

Les différents états sont les entiers de l'ensemble S= {1,2,….16}.
Quatre actions sont possibles dans chaque état :
A= {monter, descendre, gauche, droite}.
Ces actions seront représentées numériquement par :
*Aller à gauche : 1 *Descendre : 2 * Aller à droite : 3 *Monter : 4



Le but ici est de trouver le coup minimum pour que le Robot sorte de la grille c'est-à-dire qu'il arrive aux cases 16 ou 15.

Comment implémenter cette situation sous scilab ?

Les matrices Transprob (i)

i€A, (j,k)€S*S
La matrice Transprob(i) est la matrice dont l'élément Transprob(i)(j,k) représente la probabilité étant à l'état j de passer à l'état k ayant exercé l'action i.
Remarquons que pour construire cette matrice, on initialise tous ces éléments à 0 avant d'écrire les valeurs des éléments non nuls.
Ainsi nous avons obtenu [confère annexe pour tous les résultats] par exemple :

```
TransProb(1)(1,15)=1 : la probabilité étant à l'état 1 d'être à l'état 15
après avoir réalisé l'action 1(Gauche) est 1
TransProb(2)(2,6)=1 : la probabilité étant à l'état 2 d'être à l'état 6
après avoir réalisé l'action 2(Descendre) est 1
TransProb(3)(4,5)=1 : la probabilité étant à l'état 4 d'être à l'état 5
après avoir réalisé l'action 3(Droite) est 1
TransProb(4)(2,3)=1 : la probabilité étant à l'état 2 d'être à l'état 3
après avoir réalisé l'action (Monter) est 0
```

Les matrices Rewards (i)

i€A, (j, k)€S*S

La matrice Rewards(i) est la matrice dont l'élément Rewards(i) (j, k) représente la récompense (ou la sanction) accordée au robot lorsque étant dans l'état j et ayant exercé l'action i il se retrouve dans l'état k. En réalité le robot a une récompense (+1) dans la seule situation où il atteint le but sinon il a une sanction (-1) et si l'action est impossible il a alors une récompense nulle (0). Bien évidement on initialise tous les éléments de la matrice à 0 avant d'écrire les valeurs des éléments non nuls.

Ainsi nous avons obtenu [confère annexe pour tous les résultats] par exemple :

```
Rewards(1)(1,15)=1 : La récompense obtenue par le robot qui étant dans
l'état 1 ayant exercé l'action 1 se retrouve dans l'état 15 est 1
Rewards(2)(1,5)=-1 : La récompense obtenue par le robot qui étant dans
l'état 1 ayant exercé l'action 2 se retrouve dans l'état 5 est -1
Rewards (3)(14,16)=1 : La récompense obtenue par le robot qui étant dans
l'état 14 ayant exercé l'action 3 se retrouve dans l'état 16 est 1
Rewards (4)(4,16)=0 : La récompense obtenue par le robot qui étant dans
l'état 4 ayant exercé l'action 4 se retrouve dans l'état 15 est 0
```

La matrice Actions-States

(i,j)€S*S

C'est la matrice dont les éléments Actions_States(i,j) ont la valeur %F si le passage de i à j est impossible et %T si le passage de i à j est possible par une action des quatre actions. Nous avons obtenu [voir annexe pour tous les résultats] par exemple : Actions_States(1,5)=%T car le robot peut passer de l'état 1 à l'état 5 en réalisant l'action 2. Actions_States(1,16)=%F car le robot ne peut jamais passer de l'état 1 à l'état 16 en une action.

A la fin de notre implémentation sous scilab nous avons la grille des mouvements suivant :

Sortie 2			
			Sortie 1

La matrice Pi

La matrice Pi représente la politique optimale à suivre par le robot pour chaque état. C'est une matrice de 16 colonnes et d'une ligne qui a pour élément Pi (1, i)=j où i est l'état dans lequel se trouve le robot et j l'action optimale réalisée par le robot.

On peut alors schématiser les mouvements optimaux du robot par :

Sortie 1	⇦	⇦	⇦
⇧	⇦	⇦	⇩
⇧	⇨	⇨	⇩
⇨	⇨	⇨	Sortie 2

III) Deuxième exemple : The Gambler

Un joueur va parier sur les résultats de la séquence de plusieurs lancers d'une pièce de monnaie. Si la pièce tombe sur le côté visible face, alors il gagne autant d'argent qu'il a misé sur le lancer, mais si la pièce tombe sur le côté visible pile, il perd sa mise. Le jeu se termine quand le joueur gagne 100 $, ou perd tout son argent. A chaque lancer, le joueur doit décider quel part de son capital il doit miser. Ce problème peut être formulé comme un MDP discontinu, épisodique et fini.

Un état correspond au capital du joueur, s=states $\in$ {0, 1, 2, …, 100}, il y a 101 états. Les actions correspondent aux mises du joueur avant chaque lancer, a=actions $\in$ {0,1,…,min(s, 100-s)}, il y a au plus 51 actions, ce qui signifie que le joueur ne peut parier que 50 $ au plus. Par exemple, si le joueur a un capital de 51 $, il pourra parier 49 $ pour atteindre 100 $.

La récompense est 0 pour tous les couples (états, actions) sauf si le joueur atteint son but, la récompense sera alors 1. La fonction d'évaluation donne la probabilité de gagner à chaque état. Une politique serait une association des niveaux du capital à parier.
La politique optimale maximise la probabilité d'atteindre le but. On suppose que la probabilité que la pièce tombe sur face p soit égale à 0,4.

Les transprobas représentent la transition d'un état s à un état s' suivant une action a. Ce sont des matrices M(101,101), il y a 51 transprobas (car 51 actions au maximum).

Ex : la transproba(1) représente l'action : le joueur mise 0 $ est la matrice identité.
la transproba(2) représente l'action : le joueur mise 1 $,

transproba(2) =

	états finaux
états initiaux	[0……0 ;
	0,6 0 0,4 0……...0 ;
	0 0,6 0 0,4 0…….0 ;
	. . . . ;
	. . . . ;
	0 . . . . 0 ;
	0……...0 0,6 0 0,4 ;
	0……0]

Les récompenses sont aussi des matrices M(101,101) et sont au nombre de 51. La récompense vaut 1 si le joueur gagne 100 $ sinon 0.

Ex : rewards(2) =

[0……0 ;
. . ;
. . ;
. . ;
0……0 ;
0……..0 1 ;
0……0]

La probabilité de choix de chaque action en fonction de l'état est une matrice à paramètre T=true, F=false M(101,51).

Actions-states=

	1 . ……51
1	[TF……F ;
.	TTF……..F ;
.	. . . . ;
.	. . . . ;
.	. . F;
.	. T;
.	. . F;
.	. . . . ;
.	. . . . ;
.	TTF……;F ;
101	TF……....F]

On note que toutes les matrices seront créées avec des boucles for.
Du fait du nombre important de paramètres, l'implémentation des matrices à paramètres T et F n'a pas été possible, nous avons donc remplacé les paramètres T et F par 1 et 0.

De plus, afin de diminuer le temps de calcul de la politique optimale, nous avons décidé de réduire le nombre d'états à 21 et le nombre d'actions à 11.

IV) Troisième exemple : Jack's Car Rental

Cette partie introduit un exemple intéressant. Cependant il ne sera pas traité intégralement. L'exemple correspondant n'a pas été implémenté. Nous essayons d'expliquer le problème et de proposer une piste pour sa résolution. Ceci pourra faire l'objet d'un futur projet pour un éventuel groupe d'étudiants.
Jack possède deux garages de voitures. Chaque garage contient de 0 à 10 voitures qu'il fait louer.
Pendant la journée, les voitures peuvent sortir (être louer) ou être rapporter dans les proportions suivantes :

-Pour le garage1 : 3 voitures sortent et 3 voitures entrent. Pour chaque voiture qui sort, 10$ sont encaissés. Même si en début de journée il n'y a qu'une seule voiture, celle-ci sort et 3 voitures rentrent quand même.
-Pour le garage2 : 4 voitures sortent et 2 voitures entrent. Pour chaque voiture qui sort, 10$ sont encaissés. Même si en début de journée il n'y a qu'une seule voiture, celle-ci sort et 2 voitures rentrent quand même.

Pendant la nuit, les voitures peuvent être déplacés d'un garage à l'autre. Chaque voiture déplacée coûte 2$.
Comme pour les exemples précédents il faut poser le problème sous formes d'états, d'actions, de récompense et de pénalités.
Les différents états sont le nombre de voiture qu'il y a dans les deux garages. La numérotation que nous proposons est la suivante :
Ainsi il y a 121 états (11 x 11).

Numéro de l'état	Nombre de voitures dans le garage1	Nombre de voitures dans le garage2
1	0	0
2	0	1
3	0	2
4	0	3
5	0	4
6	0	5
7	0	6
8	0	7
9	0	8
10	0	9
11	0	10
12	1	0
13	1	1
14	1	2
15	1	3
16	1	4

…..etc.…….
jusqu'à :

111	10	0
112	10	1
113	10	2
114	10	3
115	10	4
116	10	5
117	10	6
118	10	7
119	10	8
120	10	9
121	10	10

Les actions correspondent au nombre de voitures déplacées du garage1 au garage2 pendant la nuit.
Nous avons décidé que nous pouvions déplacés 4 voitures au maximum.

Numéro d'action	Nombre de voitures déplacées de nuit
1	4
2	3
3	2
4	1
5	0
6	-1
7	-2
8	-3
9	-4

Un signe positif signifie que l'on déplace les voitures du garage1 au garage2.
Un signe négatif signifie que l'on déplace les voitures du garage2 au garage1.

Une fois les états et les actions définis il devient aisé de construire les matrices de probabilité Transprobas. Chaque composante de cette matrice est Transprobas(k) qui correspond à l'action k.

Ex : La première composante de cette matrice est Transprobas(1) qui correspond à l'action 1. Elle correspond à une matrice de 121 lignes et 121 colonnes.
Chaque ligne correspond à l'état précédent (compris entre 1 et 121). Chaque colonne correspond à l'état suivant (compris entre 1 et 121). On mettra un 1 dans la case ij, lorsque l'on sera passé de l'état i à l'état j en ayant fait l'action 1, c'est-à-dire déplacer de nuit 4 voitures. On choisit de mettre 1 car c'est un événement certain.

Ex : Supposons que nous sommes dans l'état suivant :

« 4 voitures dans le garage1 et 0 voitures dans le garage2. » (C'est l'état numéro 45)

Puis on effectue l'action 1 pendant la nuit. Au matin, on est dans l'état intermédiaire :

« 0 voitures dans le garage1 et 4 voitures dans le garage2. »

Puis, pendant la journée, 3 voitures sortent et 3 voitures entrent dans le garage1, et 4 voitures sortent et 2 voitures entrent dans le garage2. L'état final est donc :

« 3 voitures dans le garage1 et 2 voitures dans le garage2. » (C'est l'état numéro 36)

Donc, dans Transprobas(1), on mettra un 1 dans la case(ligne 45, colonne 36).

Le remplissage de cette matrice est assez long mais une récurrence semble apparaître sur la façon dont elle est remplie de même que pour les autres composantes Transprobas(k).

Bibliographie

[1] Richard S. Sutton, Andrew G. Barto. « Reinforcement Learning (Adaptive Computation and Machine Learning). Ed. MIT Press, Cambidge, MA, 1998.
[2] http://www-anw.cs.umass.edu/~rich/book/the-book.html