DEA Sciences Cognitives
Année1991-1992

Sujet :

Q-learning et apprentissage par renforcement

Pierre SAUREL

Rapport de stage DEA Sciences Cognitives
Directeur : Paul Bourgine

Avril-Août 1992

CENTRE NATIONAL DU MACHINISME AGRICOLE, DU GENIE RURAL, DES EAUX ET DES FORETS
Groupement d'Antony
Division : *Électronique et intelligence artificielle*
BP 121, 92185 ANTONY CEDEX, Tél. : (1) 40 96 61 21 ; Télex : 632 672 F ; Télécopie : (1) 40 96 60 80

Résumé :

Ce mémoire s'inscrit dans le cadre général de l'étude de l'apprentissage par renforcement. Je me suis plus particulièrement intéressé à l'algorithme du Q-learning.

Dans une première partie j'introduis l'algorithme à travers un bref rappel historique, en montrant les liens constatés avec la programmation dynamique.

Dans une seconde partie, je m'intéresse à l'exemple paradigmatique du problème du labyrinthe avec une seule récompense, pour lequel je démontre la convergence de l'algorithme. Ceci me permet de faire le lien avec les fonctions de Liapounov, en montrant que le Q-learning agit en construisant des fonctions de Liapounov. J'utilise ces différents résultats pour en déduire de nouveaux algorithmes.

Dans une troisième partie, j'essaye d'élargir l'utilisation de l'algorithme au problème du labyrinthe avec plusieurs récompenses. Mais les difficultés soulevées nous montrent les limites des liens existant entre Q-learning et programmation dynamique.

Dans une quatrième partie, nous introduisons quatre voies différentes permettant d'étudier plus finement comment opère le Q-learning en tant que méta-dynamique construisant des suites de fonctions de Liapounov. L'objectif est d'établir à partir de ces propriétés de nouveaux algorithmes prolongeant le Q-learning dans le cadre de l'apprentissage par renforcement. Ces algorithmes plus souples doivent pouvoir s'appliquer à d'autres classes de problèmes que celle du labyrinthe avec une unique récompense.

Remerciements :

Je tiens à remercier mon directeur de stage, M. Paul Bourgine, pour ses directives et commentaires qui m'ont permis de mieux comprendre les mécanismes délicats du Q-learning.
Je remercie Hugues Bersini qui a su me faire profiter de son expérience et m'a exposé clairement les problèmes posés par l'apprentissage par renforcement.
Je remercie G-H Portefait qui a réalisé les interfaces graphiques et m'a aidé à implémenter l'algorithme et certains de ses prolongements.
Je remercie M. Clément responsable de l'option Mathématiques Appliquées pour son aide et sa disponibilité tout au long de l'année.
Je remercie enfin tout les membres du laboratoire d'Intelligence Artificielle du CEMAGREF, et plus particulièrement J. Patinel et R. Munos qui m'ont aidé tout au long de mon stage.

SOMMAIRE

INTRODUCTION :

J'ai effectué mon stage de fin d'études au CEMAGREF (Centre National du Machinisme Agricole, du Génie Rural, des Eaux et des Forêts) dans le laboratoire d'Intelligence Artificielle animé par Paul BOURGINE. Le CEMAGREF conçoit et développe des outils pour répondre aux besoins de l'agriculteur et des industries agro-alimentaires. Il a aussi pour mission l'amélioration et la valorisation des ressources naturelles.

J'ai travaillé plus particulièrement avec Paul Bourgine et Hugues Bersini, chercheur à l'Université Libre de Bruxelles, qui occupe actuellement un poste à mi-temps au CEMAGREF.

Dans ce cadre mon sujet a été l'étude de l'apprentissage par renforcement. L'intérêt est de concevoir des systèmes autonomes, par exemple des robots sans pilote capables d'apprendre à atteindre certains objectifs. Pour cela, les robots ont des capacités cognitives qui leur permettent d'avoir accès à leur environnement local. De plus, ils obtiennent une récompense qualitative lorsqu'ils atteignent leur objectif, ce qui a lieu généralement après un délai important.

Le problème relève en fait de deux domaines qui apparaissent extrêmement liés dans les applications réelles : l'apprentissage, si le processus est répété et la capacité d'adaptation lorsque l'environnement se modifie. Rapidement il s'avère que la notion de renforcement est nécessaire pour "propager" une récompense lorsque l'on cherche à faire de l'apprentissage non supervisé pour ce type de problèmes. A la lecture d'articles et suite à des conversations avec Paul Bourgine et Hugues Bersini, il nous a semblé que la voie la plus prometteuse pour résoudre notre problème était celle ouverte par le Q-learning.

PREMIERE PARTIE :

ORIGINE ET

POSITION DU PROBLEME

1.1 Les problèmes d'apprentissage par renforcement

Barto et Sutton ont les premiers abordé et étudié le problème et l'algorithme que nous allons présenter ici. Ils se placent dans la tradition behavioriste et reprennent le modèle comportemental de Rescorla et Wagner [26] sur l'étude des souris dans un labyrinthe.

Un des objectifs de ce travail est de trouver un algorithme efficace qui permette d'exploiter l'apprentissage par renforcement. Celui-ci possède deux caractéristiques principales. Tout d'abord, l'information reçue est très pauvre. De plus, un délai temporel intervient car les actions ne sont validées que lorsque l'on a atteint la cible.

Dans ce cadre, les difficultés à surmonter proviennent essentiellement du fait que le système ne possède pas de modèle de lui-même et qu'il n'a pas connaissance du comportement idéal. On ne possède donc pas d'output idéal et il est impossible d'établir exactement quelle doit être l'erreur à minimiser. On ne peut pas utiliser une descente de gradient directement.

1.2 L'article de départ, l'ASE et l'ACE (1983)

Dans leur article Barto et Sutton [1] cherchent à résoudre le problème du pendule inverse grâce à deux éléments qui coopèrent et obtiennent un renforcement lorsque le pendule tombe.

Le premier élément, ASE (Associative Search Element) choisit les actions et en particulier la force à appliquer au pendule en fonction de l'espérance de vie de ces actions. L'ASE retient la dernière action choisie et compte son temps de vie, c'est à dire le nombre de coups après cette action jusqu'à ce que le pendule tombe.

Le second élément, ACE (Adaptative Critic Element) reçoit un renforcement du monde réel lorsque le pendule tombe ou lorsqu'il rencontre les butées du système. Il calcule alors par prédiction la valeur d'un renforcement interne local qu'il transmet à l'ASE.

Il est important de noter la complémentarité de ces deux éléments. De plus dans cet article l'ASE et l'ACE sont implémentés sous forme de réseaux de neurones. Les poids sont remis à jour comme suit :

- pour l'ASE, $w_i(t+1) = w_i(t) + \alpha\rho(t)e_i(t)$

avec $e_i(t)$ l'éligibilité de l'entrée i à l'instant. L'éligibilité est fonction du produit de l'espérance de vie, l'entrée et la sortie.L'éligibilité représente une estimation de l'intérêt que l'on a de choisir une action. $w_i(t)$ correspond aux poids du réseau à l'instant t, $\rho(t)$ le renforcement estimé par l'ACE, α un réel qui permet une pondération des modifications.

- pour l'ACE, $v_i(t+1) = v_i(t) + \beta\rho(t)x_i(t)$

et $\rho(t) = r(t) + \gamma p(t) - p(t-1)$

où $p(t) = \Sigma\, v_i(t)x_i(t)$ est la prédiction de la durée de vie de l'action à l'instant t

avec $x_i(t)$ l'entrée i à l'instant t, r(t) le renforcement réel à l'instant t et γ un facteur horizon, réel compris entre 0 et 1 qui fait que l'on tient moins compte de p(t) qui va arriver que de p(t-1) qui a déjà eu lieu.

Dans cet article de nombreuses idées sont mélées, l'utilisation des réseaux, l'idée de renforcement réel ou estimé, la prédiction d'une durée de vie. Ces différents thèmes qui seront analysés par la suite séparément. Nous en retiendrons un essentiellement, l'analyse des modifications à effectuer pour établir la valeur des prédictions.

1.3 Temporal Difference (TD)

L'article qui précède est extrêmement riche et était très novateur pour l'époque. Par la suite Barto et Sutton ont étudié plus particulièrement comment modifier les poids du réseau. Ils ont abouti à un algorithme indépendant des réseaux qui peut s'appliquer pour une classe de problèmes mieux précisée.

Un agent doit maximiser le total des récompenses r(t) qu'il reçoit du monde. Le monde est considéré comme un espace S. A chaque état x de S on attribue des valeurs V(x), qui correspondent à la récompense attendue si on se donne un comportement optimal. V(x) sera appelée la qualité attribuée à l'état x. Pour tout x appartenant à S, on souhaite que :

$$V(x) = \max r(x,a) + \gamma V(Suc(x,a))$$

a est l'action choisie en x et Suc(x,a) est l'état de S obtenu lorsque l'on applique a en x.

$0< \gamma <1$ est le facteur de décompte ou facteur "horizon", il correspond au taux de réactualisation en économie. On multiplie par un facteur γ les valeurs estimées pour l'année qui suit lorsque l'on en tient compte pour l'année courante. De cette manière on pondère les valeurs par le risque d'erreur lié à l'éloignement de la prédiction.

On en déduit alors l'algorithme suivant de remise à jour de V(x) que l'on appellera **Temporal Difference** :

$$V(x) \leftarrow V(x) + \alpha(R(x,a) + \gamma V(Suc(x,a)) - V(x))$$

Il reste alors à choisir les actions lorsque l'on se trouve à l'état x, ce qui nous permet d'obtenir y = Suc(x, a), successeur de x lorsque l'on fait l'action a. Deux cas sont alors envisagés :

- dans le cas déterministe, on choisit le successeur de x comme suit :

choix = { y/ V(y) = max V(z)}

avec z successeur possible de x

-dans le cas probabiliste, on choisit l'action avec une loi de probabilité qui dépend de V(x). On prend classiquement une distribution de Boltzmann sur les successeurs possibles de s.

Le but est que dans chaque état x, V(x) à l'instant t converge vers

$$\sum_{k=0}^{\infty} \gamma^{k} r(t + k + 1)$$

Dayan a donné des exemples d'utilisation de TD [3].

1.4 La programmation dynamique (DP)

En 1986, Barto et Sutton se sont rendu compte des points communs qui existaient entre TD et la programmation dynamique [2]. Aussi nous présentons ici succintement la programmation dynamique ce qui nous permettra de porter un nouvel éclairage sur l'algorithme [14], [16].

Pour pouvoir résoudre un problème par la programmation dynamique, trois conditions sont nécessaires :

(i) le problème est divisé en étapes qui nécessitent toutes une prise de décision.

(ii) chaque étape possède différents états

(iii) l'effet d'une prise de décision à chaque étape est la transformation de l'état courant en un état dépendant de l'étape suivante

Si les conditions précédentes sont vérifiées, on a alors le **principe d'optimalité :**
l'état courant étant donné, une politique optimale pour les étapes restantes est indépendante de la politique adoptée dans les étapes précédentes.

La procédure de résolution opère alors par chaînage arrière étape par étape. Une relation récursive relie la politique optimale au niveau n avec celle au niveau n+1

1.5 Liens entre Temporal Difference et la programmation dynamique

Temporal Difference cherche à maximiser

$$\sum_{k} \gamma^{k} r(k)$$

Ceci correspond à un critère usuel en programmation dynamique.

Cependant contrairement aux problèmes résolus par Temporal Difference, la programmation dynamique possède une connaissance totale des différentes étapes et des états. Avec TD on doit faire de l'exploration pour découvrir les étapes et les états. Ensuite TD construit une solution à partir de cette connaissance partielle comme s'il connaissait l'espace dans son entier.

Dans cette mesure, si la solution obtenue par TD converge vers la solution de la programmation dynamique, elle est également la solution optimale du problème que l'on cherchait à résoudre.

1.6 De TD au Q-learning

Lorsque Watkins développe le Q-learning en 1989, son idée directrice est d'essayer de lier une estimation des actions et les qualités des positions V(x) [13]. Il va

donner des qualités à des case-actions W_{xa}. Watkins fait une remise à jour de case-actions, qui comprennent ainsi les qualités des cases et l'estimation du résultat des actions.

On a alors $W_{xa} <- W_{xa} + \alpha(r(y) + \gamma W_{yb} - W_{xa})$
avec y l'état dans lequel on se trouve lorsque l'on a fait l'action a en l'état x, et b l'action choisie en y.

Dans le cas déterministe, Watkins affirme que les qualités ainsi modifiées convergent, en comparant le Q-learning à la programmation dynamique. Cependant nous n'avons trouvé nulle part une démonstration de ce fait.

Lorsque l'on fait un choix probabiliste des actions on prend classiquement un tirage suivant une distribution de Boltzmann.

Soit a' qui décrit l'ensemble des actions possibles en x,

$$P(a/x) = \frac{e^{(W_{xa})}}{\sum_{a'} e^{(W_{xa'})}}$$

En fait le problème que doivent résoudre ces deux stratégies est de trouver un équilibre entre exploitation et exploration. En effet dans le cas déterministe, puisque l'on prend le maximum, la phase d'exploration pendant laquelle seules quelques cases ont des qualités non nulles est limitée aux premières itérations et ensuite le robot autonome, ou une souris artificielle fait toujours le même choix, celui de la qualité maximale.

Plus particulièrement nous travaillerons sur le cas d'une souris artificielle cherchant une sortie dans un labyrinthe comportant des obstacles. La souris peut faire quatre actions : aller en haut, en bas, à gauche et à droite.

Nous travaillerons avec l'hypothèse cognitive suivante : la souris possède une vue de son environnement limitée aux quatre cases qui lui sont adjacentes. En particulier à chaque instant, la souris a uniquement accès aux qualités qu'elle attribue aux case-actions de la case où elle se trouve et de celles qui sont adjacentes.

DEUXIEME PARTIE :

LE PROBLEME
DU LABYRINTHE
AVEC UNE RECOMPENSE

Introduction

Dans toute la suite de ce rapport, nous présenterons les différents résultats que nous avons obtenus au cours du stage. De fait le problème étudié était pluridisciplinaire, puisque l'algorithme s'appuie sur un modèle provenant de la psychologie [26]. Notre travail est surtout théorique. Il a été guidé par une étude mathématique, mais aussi par la nécessité de pouvoir implémenter les différents algorithmes que nous avons déduits. Certains de ces algorithmes nous ont été suggérés par des résultats expérimentaux sur les rats [23], [24], [25].

Nous avons essentiellement étudié les valeurs numériques des qualités et leurs propriétés mathématiques. Ainsi notre intérêt s'est porté sur un point rarement discuté dans les articles dont nous avons pu disposer. Dans ceux-ci en effet, la question est toujours de savoir si oui ou non la tâche a été apprise, ou si elle est accomplie [4]. Notre travail constitue une première étude théorique du Q-learning. Grâce à l'étude des propriétés mathématiques des qualités et en particulier leur convergence, nous pouvons préciser un domaine d'utilisation pour lequel l'algorithme est convergeant. Cette étude nous a également permis de définir convenablement des critères d'apprentissage.

Nous sommes partis de l'exemple le plus simple d'utilisation du Q-learning, le problème du labyrinthe pour lequel nous avons essayé d'étudier les propriétés de convergence des qualités lorsqu'on les modifie par Q-learning.

2.1 Le problème du labyrinthe

a) en dimension 2

On place une souris dans un labyrinthe, et elle doit réussir à trouver la sortie.
Plus précisément, on se donne un espace d'état discrétisé, et un objectif fixé dans l'une des cases de notre espace. Généralement, la discrétisation se fait sous forme de carrés tous de même taille. Puis on donne à la souris des possibilités de se déplacer dans le labyrinthe. Habituellement, la souris peut se déplacer d'une case vers la droite, la gauche, le haut et le bas mais pas suivant les diagonales. Ceci définit pour une case l'ensemble de ses successeurs possibles. De plus des obstacles sont placés dans le labyrinthe ce qui réduit l'ensemble précédent. On suppose de plus que de toute case, il existe un chemin qui mène à la sortie.

La souris associe à chaque case des case-actions, une pour chaque action qu'elle peut effectuer. Au départ toutes les qualités des case-actions sont initialisées à 0.

La souris est alors placée dans le labyrinthe en un point quelconque d'entrée, elle choisit une action, se déplace puis arrive dans une nouvelle case, modifie la qualité de la case-action précédente par Q-learning, ceci jusqu'à ce qu'elle atteigne l'objectif. Là elle reçoit la récompense, modifie la qualité de la case-action précédente ainsi que celle de l'objectif. Ensuite on replace la souris en un point quelconque du labyrinthe qui sera sa nouvelle entrée, on laisse l'objectif toujours au même endroit, et la souris recommence à chercher la sortie, de la manière décrite ci-dessus.
On dira que la souris a réalisé un pas lorsqu'elle s'est déplacé d'une case, et qu'elle a réalisé une itération lorsqu'elle a trouvé la sortie.

Il s'agit bien d'un problème d'apprentissage, puisque l'on a répétition de la tâche à réaliser. La souris reçoit un renforcement, la récompense à la sortie, mais l'apprentissage est non supervisé car au milieu du labyrinthe aucun observateur extérieur ne donne d'information à la souris qui ne peut effectuer ses choix que d'après la carte, la "représentation" qu'elle s'est construite.

b) le problème du labyrinthe généralisé

On peut généraliser le problème comme suit. On se donne un graphe composé d'un sous-ensemble A de Z^p, et d'une relation $\Gamma \subset A \times A$ qui fournit l'ensemble des successeurs d'un noeud. Soit C_f un point de A correspondant à la sortie. Le problème du labyrinthe généralisé consiste à trouver une suite de fonctions Γ_q qui mènent à C_f. On fait de plus l'hypothèse que de tout point, on peut trouver un chemin qui mène à la sortie. On la traduit par :

$$\bigcup_q \Gamma_q^{-1} \supset A$$

Le problème du labyrinthe généralisé est celui effectivement étudié dans notre travail. Cependant pour plus de clarté, afin de pouvoir visualiser les situations et expliciter les hypothèses cognitives que nous avons pu faire, nous nous réfèrerons au problème en dimension 2. Ceci n'ôte rien pour autant à la généralité de notre propos et de nos raisonnements.

2.2 Aspects théoriques

Nous nous intéressons ici au Q-learning déterministe avec un seul point où l'on donne une récompense, la sortie du labyrinthe. De plus cette récompense est positive et constante au cours du temps puisqu'elle vaut 1 à chaque fois que l'on atteint l'objectif.

Nous dirons que le Q-learning est à un pas lorsque le choix de l'action se fait à partir des qualités des case-actions de la case où la souris se trouve. Le Q-learning à deux pas se définit par un choix d'action à partir des case-actions des cases qui sont visibles par la souris, c'est à dire des quatre cases adjacentes à celle où la souris se trouve.

définition : On dira que la fonction Q de C dans C qui à chaque case-action associe sa qualité est bien stratifiée ssi

$$\forall c \in C, \{ c' \in \{Suc(c)\} / Q(c) \leq \gamma Q(c') \} \neq \emptyset$$

Le choix des actions se fait selon la stratégie du max. Cependant nous élargissons ce choix grâce à la propriété de croissance de la valeur actualisée (CVA) que nous rappelons ci-après.

propriété CVA : soit γ un nombre entre 0 et 1, soient c et c' deux case-actions, c' étant un successeur possible de c. Soit Q la fonction qui à une case-action associe sa qualité,

c et c' vérifient la propriété CVA ssi $Q(c) \leq \gamma Q(c')$

f : C->C vérifie la propriété CVA en c ssi c et f(c) vérifient la propriété CVA

f est CVA par rapport à Q ssi f vérifie la propriété CVA en tout point $c \in C$

Par la suite nous dirons CVA, sans préciser que c'est par rapport à Q.

Notre démonstration (voir Annexe 1) établit que le successeur de c dont la qualité est maximale vérifie la propriété CVA avec c.

Le résultat théorique principal que nous avons obtenu est le suivant: le Q-learning est un algorithme qui fournit une solution pour le problème du labyrinthe à un objectif, pour laquelle les qualités de toutes les case-actions sont convergeantes en fin d'apprentissage. De plus les solutions construites jusqu'à la sortie sont des chaînes sans boucle, c'est à dire des chemins. Les qualités le long de ces chemins ont pour valeur $1/\gamma^n$ avec γ réel, et n longueur du chemin jusqu'à la sortie. Le long de ces chemins, le Q-learning converge donc vers la solution de la programmation dynamique.

La formulation la plus élégante de notre résultat s'énonce en termes de fonctions de Liapounov [17].

définition : Soit $F : R^n \to R^n$ un difféomorphisme, $V : R^n \to R$ est une fonction de Liapounov pour F centrée en p ssi
1) $V(x) > 0$ pour $x \neq p$
2) $V(p) = 0$
3) $VoF(x) \leq V(x)$ avec égalité ssi $x = p$

On dira de même que V est une fonction γ-Liapounov si dans la définition précédente, le 3) s'écrit $\gamma VoF(x) \leq V(x)$.

Dans notre problème, on a le maximum de V en p, $V(p) \leq 1$ mais au lieu d'avoir décroissance le long de la dynamique, on a croissance. Nous conservons le même vocabulaire.

F qui est la dynamique, correspond au choix du successeur dans l'algorithme du Q-learning.

Notons que l'on obtient comme corollaire des définitions précédentes :

Si Q est bien stratifiée, c'est une fonction γ-Liapounov pour toute dynamique F CVA

On peut alors formuler la proposition 2 de l'Annexe 1 comme suit :

Proposition 2 bis :

Soit *T selon nos notations de la démonstration, l'opérateur qui modifie les qualités des case-actions.*

C_n l'ensemble des case-actions de l'espace d'état dont la qualité est non nulle après la n-ième itération.

Q_n l'application qui associe à chaque case-action de C_n sa qualité. Donc Q_{n+1} est le modifié par T de Q_n

F_n l'application qui à chaque point de C associe son successeur au cours de la n-ième itération.

Si Q_n est bien stratifiée, alors pour tout c ∈ C et toute dynamique F_n CVA pour Q_n partant de c, Q_{n+1} est bien stratifiée.

A chaque itération, c'est à dire quel que soit n, Q_n est une fonction γ-Liapounov pour F_n, et même pour tout choix de successeur qui conserve la propriété CVA, donc pour F_k, quel que soit $k \geq n$.

T est alors un opérateur de construction de fonctions γ-Liapounov.

On obtient une suite de fonctions de γ-Liapounov (Q_n).

La suite (C_n) est croissante au sens où $C_n \subset C_{n+1}$. C_n incorpore de nouveaux éléments, et si l'on explore tout C, on aura même un N_0 tel que pour tout $n \geq N_0$, $C_n = C$.

De plus la suite Q_n est convergeante vers Q, fonction de γ-Liapounov pour au moins une dynamique F. Pour toute dynamique F possible pour Q, c'est à dire qui conserve la propriété CVA, Q donne comme qualité à chaque case-action, celle que donnerait la programmation dynamique en suivant la même dynamique.

Dans la suite nous dirons qu'une fonction est Liapounov pour γ-Liapounov.

2.3 Interprétation des résultats

Il est important de noter l'aspect géométrique de la démonstration de la convergence du Q-learning à un pas. A partir d'une récompense ponctuelle, le Q-learning construit une surface de qualités strictement positives, possédant un sommet, là où la récompense est établie. Progressivement le Q-learning étend cette surface à tout l'espace.

Cette surface est subjective et dépend des capacités cognitives du sujet dans le labyrinthe. En effet ce sont ses capacités cognitives qui lui permettent de définir les successeurs possibles (fonction Suc avec les notations de la démonstration). Du point de vue cognitif les différentes valeurs attribuées à chaque case sont soit des traces laissées physiquement comme le fait le rat, soit des paramètres modifiés d'une représentation mentale.

Le Q-learning a pour effet la construction de lignes de niveau. Le robot se déplace sur des trajectoires construites passant d'une ligne de niveau à la suivante selon une stratégie définie soit par le choix du max, soit par le choix d'un des éléments qui vérifient le propriété CVA. Le robot construit par Q-learning des fonctions de Liapounov pour la dynamique du choix du successeur. Lorsque le robot a suffisamment appris, il se déplace alors le long de trajectoires qui vérifient la propriété CVA.

Un des aspects importants de notre démonstration est qu'elle s'effectue dans R^n. Ainsi notre espace d'état n'est plus nécessairement lié à l'espace physique. Ceci nous libère d'un problème cognitif important car il est nécessaire pour cet algorithme de conserver une matrice correspondant à l'espace d'état. Ceci implique soit un marquage physique de l'espace d'état ce qui est le cas pour les rats, soit la conservation mentale d'une matrice de l'espace physique. Cependant si le labyrinthe est grand l'hypothèse de stockage d'une carte mentale est peu vraisemblable. En travaillant sur un espace d'état non nécessairement lié à l'espace physique, le problème disparaît.

Il suffit de construire un espace d'état en prenant des paramètres d'état "pertinents" pour le problème que l'on cherche à résoudre, par exemple l'odeur ou la couleur. C'est dans cet espace lié à des variables perceptives que le Q-learning attribue des qualités. Plus précisément, on peut par exemple choisir en abscisse des intensités d'odeur discrétisées de 0 à 10 et en ordonnée des couleurs de 0 à 5, la souris se déplace alors dans l'espace physique, mais à chaque fois qu'elle doit prendre une décision pour se déplacer, elle observe son environnement ce qui la place dans une certaine case de son espace perceptif odeur-couleur, pour cette case elle possède des case-actions avec différentes qualités qui lui permettent d'effectuer son choix d'action. Elle modifie également les qualités de ces case-actions par Q-learning. Il faut noter que ceci n'est possible que si l'espace possède certaines régularités : il faut que l'espace d'état ait été convenablement choisi pour qu'en deux points différents de l'espace physique où l'on a la même observation, l'action choisie résultant de cette observation soit valide pour ces deux points.

Enfin, il est intéressant de remarquer que le sujet à chaque instant ne fait appel qu'aux données de la matrice liées aux actions possibles dans la case où il se trouve actuellement. Nous travaillons bien avec notre hypothèse cognitive de vision à une case. Ainsi l'algorithme peut correspondre à un problème d'apprentissage avec modification de l'évaluation de ce que l'on perçoit.

2.4 Liens entre Dyna et le Q-learning à un pas

a) aspects théoriques

Dyna est un algorithme qui utilise la même règle de modification des qualités que le Q-learning, mais les qualités sont attribuées à chaque case et non à des case-actions. De plus avec Dyna la souris effectue un choix sur les cases qui sont autour de la case où elle

se trouve comme pour le Q-learning à deux pas. Dyna fait alors le choix de l'action supposée connue qui la mènera dans la case choisie. Plus précisément, Dyna attribue des qualités aux cases et choisit parmi les qualités des cases autour alors que le Q-learning attribue des qualités aux case-actions mais effectue son choix à partir des valeurs dans la case où il se trouve.

Notre démonstration de la convergence du Q-learning à un pas peut facilement s'adapter pour montrer la convergence de Dyna. Pour cela, il suffit de ne pas particulariser une dimension supplémentaire correspondant aux actions. Les p dimensions correspondent donc toutes à un espace d'état qui ne comprend pas les actions. Puis on modifie la définition 2 du successeur en remplaçant simplement p-1 par p puisque l'espace des actions a été supprimé.

b) les différences

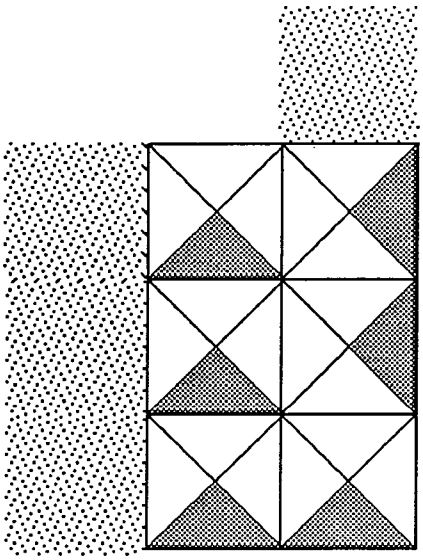
Ces deux algorithmes sont équivalents si quand on applique une action, on sait dans quelle case on va arriver, ce qui est le cas pour le problème du labyrinthe. Cependant le Q-learning a l'avantage d'être utilisable sans la connaissance de la fonction f qui à une action associe sa case d'arrivée. Dans la mesure où l'on attribue des qualités en même temps aux actions, peu importe si la fonction f est déterministe ou si elle suit une loi de probabilité.

Pour un certain nombre de problème où cette fonction f est inconnue, le Q-learning semble avantageux par rapport à Dyna utilisé avec une estimation de f. Un exemple est donné par R. Munos [9].

2.5) Les problèmes de convergence du Q-learning à deux pas

Nous avons simulé le Q-learning à deux pas, ce qui nous a permis de constater des phénomènes inattendus qui nous permettent de dire que le Q-learning à deux pas n'est pas convergent.

a) Le premier phénomène constaté est la possibilité qu'une qualité d'une case-action soit décroissante au cours d'une itération ce qui est contraire aux propriétés des qualités démontrées dans le cadre du Q-learning à un pas. La propriété essentielle du Q-learning est qu'il conserve la monotonie des qualités entre des cases successives. Cette propriété nous a permis de démontrer la convergence du Q-learning à un pas; pour le Q-learning à deux pas, nous allons montrer que l'on peut enclencher une décroissance, le Q-learning perpétuera alors cette décroissance.



Les cases remplies de points correspondent à des obstacles.

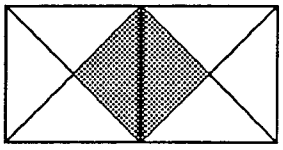
Chaque case est découpée en quatre triangles, chacun correspondant à une case-action. La pointe du triangle définit la direction où mène l'action. Par exemple, le triangle inférieur correspond à une action qui mène vers le haut. Nous utiliserons les notations suivantes : h (resp b, d, g) désigne l'action qui mène vers le haut (resp bas, droite, gauche). Par exemple Q_{4g} désigne la qualité de la case-action de la case 4, action vers la gauche. Quand on parlera de valeur d'une case, il s'agira toujours de la plus grande valeur des case-actions de cette case et on la notera Q(n° de la case). Les triangles grisés correspondent à des case-actions dont les qualités sont non nulles.

On suppose initialement que la case 2 est nulle, et que les cases 1, 3, 4, 6 sont valuées non nulles, elles vérifient toutes la propriété CVA pour la chaîne 6-4-3-1 comme indiqué sur la figure. On suppose de plus que $Q(4) \geq Q(5)$.

Puis il se trouve que la case 2 est choisie plusieurs fois comme case initiale jusqu'à ce qu'elle obtienne une valeur $Q(2) \geq Q(3)$.

Ensuite, 6 est tirée comme case initiale. On choisit alors d'aller en 4, parce que Q_{4g} est supérieure à Q_{5h}. Puis en 4, on choisit l'action h car $Q(2) \geq Q(3)$. On modifie alors Q_{6h} par Q-learning en fonction de Q_{4h} alors que c'est Q_{4g} qui nous a permis d'effectuer le choix d'aller de 6 en 4. Q4h étant plus faible que Q_{6h} puisque nul, on a Q_{6h} qui décroît. On observe bien un phénomène de décroissance de la qualité Q_{6h}.

b) Le second phénomène constaté est la possibilité pour la souris de se bloquer entre deux cases. En effet suite à des décroissances des qualités du type de celle explicitée précédemment, on peut arriver à la situation suivante :



Q_{2d} est la qualité maximum des voisines de la case 1. Donc de la case 1, on choisira l'action d dans la case 1 et on modifiera Q_{1d}. Puis Q_{1g} est la qualité maximum des voisines de la case 2. Donc on choisit l'action g dans la case 2 et Q_{1d} sera donc modifiée en fonction de Q_{2g} . Puis après application de l'action g dans la case 2, on se retrouve en 1 et on modifiera Q_{2g}. Or comme Q_{2g} et Q_{1d} étaient initialement nuls, ils le

restent, et comme on ne modifie ni Q_{2d}, ni Q_{1g} ils conservent leur propriété de maximum. Finalement la souris reste bloquée entre les cases 1 et 2 oscillant de l'une à l'autre. On peut décrire le phénomène comme suit : on choisit une case pour son optimum, mais ensuite, quand on se déplace dans cette case, on ne choisit pas forcément comme case-action celle qui correpond à l'optimum. D'une certaine manière on a rompu une certaine "connexité" nécessaire dans notre démonstration pour le Q-learning à un pas : le choix d'une action s'effectue à partir des qualités des case-actions, mais le déplacement a lieu vers une case. Dans la case d'arrivée on se place dans la case-action dont l'action correspond à celle qui vient d'être effectuée et non nécessairement dans celle qui a motivé notre choix initial.

c) Nous proposons de modifier légèrement l'algorithme du Q-learning à deux pas pour récupérer la propriété de "connexité" qui manquait : on va effectuer un second type de modification des qualités des case-actions, différent du Q-learning.

On choisit une case-action dans les voisines, ce qui nous définit un choix d'action, et l'on retient la valeur de la qualité qui nous a fait effectuer ce choix. On effectue l'action retenue, ce qui nous place dans une nouvelle case, puis l'on effectue le choix d'une action ce qui nous fixe une case-action dans cette nouvelle case. On effectuait alors une modification de l'ancienne case-action en fonction de la nouvelle par Q-learning. Au contraire, maintenant on modifie d'abord la qualité de la nouvelle case-action en lui donnant la valeur de la qualité retenue. Ensuite, on peut modifier la qualité de l'ancienne case-action en fonction de la nouvelle par Q-learning. L'algorithme ainsi modifié permet de conserver la "connexité", et notre démonstration du Q-learning à un pas peut alors s'adapter.

2.6 Le conflit exploration-exploitation : nouveaux algorithmes

Le Q-learning construit des chemins vers la sortie et converge vers la meilleure des solutions au sens de la programmation dynamique dans la partie de l'espace d'état qu'il a exploré. Cependant le Q-learning déterministe avec un choix du max pour successeur effectue très peu d'exploration et plus du tout lorsqu'il a trouvé une solution. Pour résoudre ce problème, nous avons défini deux algorithmes complémentaires.

Pour le Q-learning le choix du successeur se fait habituellement comme suit :

choix(Suc(a)) = max{ Q(b) avec b appartient à Suc(a)} (Q1)

a) Premières modifications, l'algorithme (Q2)

Notre premier algorithme se définit naturellement à la suite de notre démonstration de la convergence du Q-learning à un pas. Il consiste non plus à choisir comme successeur le max, mais n'importe quel élément successeur possible qui de plus vérifie la propriété CVA. Dans ce nouveau sous-ensemble de successeurs acceptables, on peut alors effectuer un choix probabiliste selon une distribution de Boltzmann :

algorithme (Q2) :

Soit B l'ensemble des successeurs de a qui vérifient la propriété CVA
P une procédure de tirage selon une distribution de Boltzmann
choix(Suc(a)) = P(Q(b)) avec b appartient à B (Q2)

Si l'on considère qu'un chemin construit qui aboutit à la sortie est une solution à notre problème, on peut dire que notre premier algorithme favorise l'exploration autour d'une solution. Il permet de construire les solutions de manière moins déterministe ce qui limite l'importance des décisions prises aléatoirement lorsqu'au début presque toutes les qualités sont encore nulles.

b) Un second algorithme (Q3)

Le second algorithme favorise la recherche de nouveaux chemins, complètement différents. Il consiste à favoriser l'exploration quand au départ on est sur une case évaluée à zéro. Par exemple on peut lorsque l'on est sur une case valuée à zéro choisir comme successeur trois fois sur quatre une case valuée à zéro plutôt que de prendre un morceau de chemin déjà construit.

algorithme (Q3) :

Soit B l'ensemble des successeurs de a qui vérifient la propriété CVA
B' ensemble des successeurs b de a qui vérifient Q(b) = 0
P une procédure de tirage selon une distribution de Boltzmann

-si Q(a)≠0 choix(Suc(a)) = P(Q(b)) avec b appartient à B
-si Q(a)=0,

- trois fois sur quatre, choix(Suc(a)) = P(Q(b)) avec b appartient à B'
- une fois sur quatre, choix(Suc(a)) = P(Q(b)) avec b appartient à B

(Q3)

Le nouvel algorithme reste convergent comme le Q-learning normal puisque l'on ne perturbe pas la construction de nos chemins, la propriété CVA restant conservée.

Ainsi, en augmentant l'exploration autour d'une solution du problème, et en cherchant de nouvelles solutions, on permet une exploration beaucoup plus grande de l'espace d'état. On peut alors espérer que la solution trouvée par le sujet, optimale dans l'espace exploré sera plus souvent la solution optimale du problème pour un observateur qui connaît tout l'espace d'état.

On peut considérer les algorithmes proposés (Q1, Q2, Q3 et Q4 par la suite) comme des tentatives pour modifier la dynamique F tout en conservant le fait que T construit toujours des fonctions de Liapounov pour ces nouvelles dynamiques.

c) test pour comparer Q1, Q2 et Q3

Pour comparer les trois algorithmes, nous proposons le protocole suivant : on construit trois représentations du même labyrinthe q_1, q_2, q_3 chacune suivant son algorithme respectif. Au départ toutes trois sont initialisées à 0. Le protocole se découpe en deux phases, une d'apprentissage et l'autre de test.

- phase d'apprentissage

On tire une série de points qui serviront chacun de point initial d'une itération pour les trois algorithmes. On lance chacun des algorithmes (Qi) à partir de ces différents points jusqu'à ce qu'ils atteignent la sortie, c'est à dire sur une itération, ce qui modifie les représentations q_i. Les résultats de convergence nous garantissent le fait qu'avec les trois algorithmes, la souris va bien atteindre la sortie.

- phase de test

On tire une série de points comme tests au hasard sur les bords du labyrinthes, ils serviront de points initiaux pour la recherche de la sortie pour les trois algorithmes. On lance alors les trois tests. On part d'une position initiale puis l'on effectue un choix du successeur à partir des règles Q1, Q2 et Q3 et des représentations q_1, q_2, q_3. Cependant, dans cette phase de test les qualités ne sont pas modifiées et les représentations ne servent donc qu'à effectuer le choix d'un chemin. Pour chaque représentation et chaque itération (arrivée à la sortie), on compte combien de déplacements ont été nécessaires. Puis on calcule le nombre moyen de déplacements nécessaires ainsi que l'écart type, et on compare ces différents résultats pour les trois algorithmes.

Il est alors intéressant d'étudier la sensibilité des différents algorithmes au nombre d'itération α lors de la phase d'apprentissage.

Ce test n'a pas été effectivement implémenté. Nous attendons cependant les résultats suivants :

• lorsque α est petit, les différences sont qualitatives, les trois algorithmes devraient avoir valué pratiquement le même nombre de case-actions à des valeurs non nulles. Cependant avec Q1, les cases valuées se répartiront plutôt le long d'un chemin, comme un long fil très étiré partant de la sortie. Avec Q2, on devrait avoir concentration de ces cases valuées autour d'un chemin plus court, soit un fil partant de la sortie moins étiré que pour Q1 mais comportant plus de ramifications. Avec Q3, les cases valuées se répartiront de manière plus uniforme comme une pelote de fils autour de la sortie.

• lorsque α est grand, les trois représentations devraient avoir pratiquement les mêmes cases valuées, avec plus de "trous", cases non valuées au milieu du labyrinthe pour Q1 que pour Q2 et Q3. Les trois algorithmes devraient avoir alors à peu près les mêmes longueurs de chemin, Q2 et Q3 pouvant être légèrement meilleurs.

Cependant les améliorations apportées par les modifications du Q-learning, si elles sont intellectuellement compréhensibles sont beaucoup plus difficiles à visualiser pour deux raisons :

• Pour pouvoir observer ces phénomènes il faut des labyrinthes suffisamment grands, ce qui prolonge énormément la recherche au hasard qui a lieu au début lorsque toutes les case-actions sont valuées à 0. En effet on augmente alors la distance moyenne de la case de départ à la sortie, et comme la recherche s'effectue au début au hasard, le nombre de coups pour arriver à sortir augmente exponentiellement en probabilité.

De plus le problème du labyrinthe sans obstacle possède la topologie du problème du chauffeur de taxi. Le nombre de chemins qui mènent à la sortie qui sont sans boucle et de longueur optimale est très important dans l'ensemble des chemins sans boucle qui mènent à la sortie. Ainsi même si Q2 et Q3 trouvent plus facilement ces chemins, Q1 en construit déjà un très fréquemment. Si on rajoute beaucoup d'obstacles, rapidement il n'y a plus qu'un chemin sans boucle qui mène à la sortie et alors le déterminisme de Q1 ne nous gêne plus. Pour valider l'intérêt des nouveaux algorithmes, il faut donc trouver une densité d'obstacle intermédiaire.

Cependant le fait qu'il soit difficile de trouver des situations intéressantes montrant l'apport de ces nouveaux algorithmes ne remet pas en cause leur intérêt. Dans de grands labyrinthes à densité d'obstacle variable, certaines régions intermédiaires valoriseront particulièrement ces nouveaux algorithmes, alors que pour les autres régions Q1, Q2 et Q3 donneront des résultats identiques après un long apprentissage. Par contre il est difficile d'évaluer cette densité d'obstacle autrement que numériquement pour pouvoir tester les trois algorithmes sur un labyrinthe possédant uniformément cette densité d'obstacle.

2.7 La capacité d'adaptation avec le Q-learning

a) cadre général

Il est important que le chemin construit par le Q-learning puisse s'adapter à d'autres situations que celles sur lesquelles on a appris [12]. Dans un cas au moins

l'adaptabilité s'obtient facilement : si l'on supprime un obstacle, on attribue alors à la case qui est désormais accessible la valeur maximum de ses voisines exceptée la meilleure. Ainsi les propriétés qui nous ont assuré la convergence de l'algorithme sont conservées, soit essentiellement la propriété CVA.

Cependant la surface construite ne reste valable pour un autre problème que si l'objectif ne s'est pas trop déplacé dans l'espace d'état. Si on travaille sur le problème du labyrinthe cela implique que la nouvelle sortie ne doit pas être trop éloignée de la précédente (voir paragraphe suivant). Ce résultat est naturel, il affirme simplement que l'on ne peut utiliser ce que l'on a appris que pour des situations proches de notre base d'apprentissage. Il serait intéressant de réussir à quantifier cet aspect.

b) un algorithme adaptatif

Nous proposons ici un nouvel algorithme qui permet de s'adapter à des situations où l'objectif a été déplacé : le lieu où l'on obtient la récompense se déplace donc au cours du temps. Le but est d'éviter lorsque l'on a déjà effectué un apprentissage sur un objectif de devoir repartir d'une représentation vide pour trouver et apprendre le nouvel objectif. Ce que l'on a déjà appris doit donc nous guider dans cette nouvelle recherche. Cependant pour que les informations que nous possédons sur l'ancien problème soit encore pertinentes pour le nouveau, il est préférable que l'ancien objectif soit proche du nouveau et qu'il se déplace suffisamment lentement par rapport au temps d'apprentissage.

Le principe de ce nouvel algorithme est simple : jusqu'à présent le Q-learning ne permettait que de grimper d'une ligne de niveau à la suivante avec la propriété CVA vérifiée. Cependant si l'objectif a été déplacé, ce chemin peut nous mener à un maximum local qui n'est pas l'objectif. Dans ce cas, il faut que notre nouvel algorithme permette à la souris de redescendre les lignes de niveau sans pour autant perdre la propriété CVA qui nous assure la convergence.

L'algorithme se définit comme suit, la souris doit suivre une politique avec un choix sur les successeurs qui vérifient la propriété CVA en suivant par exemple l'algorithme Q2 ou Q3 , mais pas un choix du max parce que lorsque l'on a un maximum local, le max ne vérifie plus la propriété CVA. Nous appellons (PROC1) cette procédure qui jusqu'à présent correspondait à toute la politique et la modification des poids du Q-learning. Lorsque la souris arrive à un maximum local, aucun successeur ne vérifie la propriété CVA. Dans ce cas la souris suit la procédure suivante (PROC2):

- elle effectue un choix aléatoire sur les successeurs selon une distribution de Boltzmann
- elle note la position où elle se trouve
- elle effectue son déplacement
- elle effectue à partir de sa nouvelle case un nouveau choix de successeur à l'exception de la case où elle se trouvait auparavant. De nouveau ce choix se fait en vérifiant la propriété CVA et si c'est impossible on reprend la procédure qui vient d'être proposée.

L'essentiel dans cette nouvelle procédure est que l'on n'a pas effectué de réajustement des qualités par Q-learning dans le cas d'un maximum local, ce qui nous a permis de redescendre selon les lignes de pente sans détruire ces pentes ce qui amènerait rapidement à des boucles comme nous avons pu le constater sur des simulations. Cependant si notre algorithme résoud certains problèmes, il n'est pas convergent pour tout labyrinthe. En fait il permet si la densité d'obstacle est faible de trouver un autre chemin proche. Si la densité d'obstacle est importante cet algorithme n'évite pas certains blocages de la souris. Le refus de rechoisir le maximum local que l'on vient de quitter peut rendre impossible l'accès à la sortie. Il est possible que nous ayons alors découpé le labyrinthe en deux composantes connexes, le seul chemin menant alors à la sortie repassant nécessairement par le maximum local précédent.

De nouvelles modifications permettent d'éviter ces problèmes. Si aucun successeur n'est possible, sauf la case que (PROC2) nous interdit puisqu'on en vient (cas d'une impasse), on peut alors créer une nouvelle routine (PROC3):
la souris lorsqu'elle ne peut que choisir la case qu'elle s'était interdite suit l'algorithme Q2 sur tous les successeurs (en effectuant les modifications des qualités). Elle risque alors de retrouver un maximum local mais elle s'interdit de revenir sur ses pas. Au cours de cette routine elle note toujours la case d'où elle vient et refuse de la reprendre, ce qui est fait naturellement par l'algorithme Q2 sauf là où se trouve le maximum local pour lequel on doit donc prendre cette précaution.

Finalement pour pouvoir profiter au mieux des connaissances déjà acquises sans perdre les propriétés de convergence, nous proposons de suivre le schéma suivant : Tant que la propriété CVA est vérifiée on utilise PROC1 qui permet de construire correctemment les lignes de niveau, si l'on arrive à un maximum local, on utilise PROC2 jusqu'à ce que l'on puisse réutiliser PROC1, sinon, si l'on arrive à une impasse, on utilise PROC3 qui nous ramène à un maximum local et dans ce cas on réutilise PROC2 puis PROC1.

Je ne suis pas certain que même cet algorithme modifié vérifie dans tous les cas les propriétés de convergence et ne bloque jamais. Une simulation numérique permettrait de nous guider pour regarder plus en détail les aspects théoriques de la question.

Cet algorithme est prometteur car il est à ma connaissance le seul dans le domaine du Q-learning qui permette une adaptation au déplacement du but dans la matrice des qualités apprises.

TROISIEME PARTIE :

LE PROBLEME
DU LABYRINTHE
AVEC PLUSIEURS RECOMPENSES

Introduction

Dans cette troisième partie, nous cherchons à prolonger l'utilisation du Q-learning au problème du labyrinthe avec plusieurs récompenses. Nous établissons à travers cette étude les limites des liens existant entre Q-learning et programmation dynamique. Ceci nous amène à redéfinir les aspects essentiels de l'algorithme du Q-learning dans le cadre de l'apprentissage par renforcement.

3.1 Q-learning et programmation dynamique (DP)

a) les points communs

Une des raisons principales de l'intérêt renouvelé pour les algorithmes par renforcement et en particulier pour le Q-learning est le lien qui a été établi avec la programmation dynamique. En particulier Sutton développe dans certains de ses articles des arguments pour montrer les ressemblances entre Q-learning et programmation dynamique [5], [6], [7].

Le Q-learning comme la programmation dynamique cherche à maximiser la somme des récompenses multipliées par un facteur horizon à la puissance du nombre de coups nécessaires avant d'atteindre cette récompense. De même Q-learning et programmation dynamique définissent une stratégie locale.

Ces arguments semblaient suffisant pour que Sutton se réfère directement aux démonstrations liées à la programmation dynamique sans chercher à les adapter au Q-learning. Watkins et Dayan ont récemment approfondi ce travail et ont adapté une démonstration de la convergence de la programmation dynamique pour montrer la convergence du Q-learning [13].

b) les différences

A la différence de la programmation dynamique, le Q-learning travaille sans connaître tout l'espace d'état. Le Q-learning ne peut donc fonctionner comme DP. Celle-ci coupe progressivement des branches de l'arbre des possibles, alors que le Q-learning effectue lui, au début, une exploration aveugle. Le Q-learning construit une représentation à partir d'une méconnaissance totale. De plus la recherche aléatoire peut être extrêmement longue éventuellement jusqu'à ce que le robot ait atteint toutes les cases de l'espace d'état, sachant qu'à chaque itération (à chaque sortie), une seule nouvelle case valuée à zéro est modifiée.

c) convergence du Q-learning avec plusieurs récompenses

Watkins et Dayan viennent de publier une démonstration de la convergence du Q-learning avec plusieurs récompenses en s'appuyant sur une démonstration de la convergence de la programmation dynamique [13]. Cependant, contrairement à notre démonstration pour une récompense, leur démonstration s'appuie sur des hypothèses qui permettent de faire un parallèle complet avec la programmation dynamique. Ils sont pour cela obligés de supposer que le robot passe sur chaque case et qu'il effectue chaque action un nombre infini de fois. Cette hypothèse est très forte et ne peut pas être réalisée en pratique parce que le Q-learning dans sa construction élimine progressivement certaines cases qui ne sont pas intéressantes. Au bout d'un certain nombre d'itérations, le robot ne passe plus du tout sur un certain nombre de cases. Avec son hypothèse, Watkins montre la convergence sur toutes les cases vers la solution de la programmation dynamique. Cependant lorsque l'on applique réellement l'algorithme seules certaines cases sont parcourues un nombre infini de fois, et c'est sur ces cases là que pour le cas d'une récompense unique nous avons montré la convergence du Q-learning.

d) les limites de l'algorithme dans le cas de plusieurs récompenses

Watkins affirme vraisemblablement avec raison que les hypothèses qu'il utilise sont les plus faibles pour pouvoir utiliser la programmation dynamique. Son résultat est alors purement théorique et ne correspond pas au Q-learning réel.

Dans le cas de plusieurs récompenses, il nous semble également qu'une démonstration de la convergence utilisant les aspects constructifs de l'algorithme est impossible de manière générale, car cela supposerait la création d'une surface possédant

deux propriétés contradictoires : dans le cas de deux objectifs, il faudrait que la surface soit croissante le long des trajectoires en vérifiant la propriété CVA, et qu'elle possède deux objectifs, c'est à dire deux sommets. Les problèmes rencontrés pour le Q-learning lorsqu'il y a plusieurs récompenses correspondent donc aux difficultés voire à l'impossibilité d'ajouter deux fonctions de Liapounov tout en conservant la croissance CVA jusqu'à l'objectif final.
A part dans le cas très particulier, équivalent au cas d'un objectif, où les récompenses pour les deux objectifs sont suffisamment différentes, ces deux conditions sont impossibles à concilier, et la démonstration provenant de la programmation dynamique ne permet pas de définir un algorithme convergeant.

3.2 Problèmes à plusieurs récompenses

a) problématique générale

On aura remarqué que notre démonstration correspond au problème avec un seul lieu pour les récompenses, à savoir la sortie. Cependant on peut être amené à vouloir résoudre des problèmes avec plusieurs objectifs et donc plusieurs récompenses ce qui correspond mieux à la formule générale de la programmation dynamique. Dans ce cas le Q-learning va créer des surfaces avec plusieurs sommets. Il semble difficile de faire en sorte que les chemins créés passent nécessairement par les deux sommets car localement un de ces deux sommets sera un maximum or la propriété CVA fait que le sujet ne peut pas descendre, sauf si l'on utilise des algorithmes modifiés du type de celui du paragraphe 2.7. Comme nous l'avons dit dans le paragraphe précédent, il nous semble difficile voire impossible d'adapter le Q-learning à des problèmes comportant plusieurs récompenses tout en gardant les mêmes buts que lorsque l'on utilise la programmation dynamique.

De même certains phénomènes "parasites" peuvent intervenir si on met des récompenses négatives à la place des obstacles qui ne sont donc plus des lieux interdits. En effet l'algorithme va propager les récompenses négatives. Supposons par exemple que tous les chemins qui mènent à la sortie passent par une case c encadrée de chaque côté par des obstacles. L'algorithme va créer des cuvettes là où se trouvent les obstacles, mais de plus ces cuvettes vont se propager en c, et au bout d'un certain temps, la case c va avoir une valeur trop faible pour que le sujet choisisse d'y passer. On a alors un phénomène de "traumatisme", et le sujet ne peut plus trouver de chemin menant à la sortie.

On comprend alors pourquoi nous avons essentiellement étudié pour l'instant la convergence de l'algorithme dans le cas d'une récompense positive et unique.

b) les solutions proposées

Les solutions proposées correspondent généralement à l'état d'esprit suivant [10], [11]. Un objectif est considéré comme objectif intermédiaire, lieu de passage obligatoire. Ceci permet de limiter la recherche aléatoire au début lorsque l'exploration est très importante grâce à une restriction de l'espace de recherche. En effet, si on prend toujours la même position initiale, l'objectif intermédiaire est placé dans le labyrinthe entre la position initiale et la sortie. Cependant cette solution doit plutôt être considérée comme la résolution d'un problème avec un point de passage obligé en chemin. L'état d'esprit en est effectivement distinct de celui de la programmation dynamique à plusieurs objectifs. Le Q-learning cherche alors à trouver une solution optimale entre l'entrée et l'objectif intermédiaire puis une solution optimale entre l'objectif intermédiaire et l'objectif final, alors que la programmation dynamique avec plusieurs récompenses chercherait une solution optimale pour le problème total entre l'entrée et la sortie, sachant que là où se

trouve l'objectif intermédiaire il y a également une récompense. Les solutions proposées en terme de sous objectif sont intéressantes, mais ne correspondent pas à la programmation dynamique à plusieurs récompenses.

c) autre proposition

Nous proposons une autre solution au problème d'un labyrinthe dans lequel une souris devrait trouver deux objectifs par exemple l'un correspondant à de la boisson l'autre à de la nourriture. On peut par exemple créer et apprendre selon deux représentations mentales correspondant chacune à l'un de nos besoins, l'une pour la soif, l'autre pour la faim, et la recherche se ferait soit dans l'une des représentations mentales soit dans l'autre selon des paramètres internes fixant si la faim ou la soif sont l'objectif prioritaire de la recherche.En dessous d'une certaine valeur d'un paramètre de besoin en eau, c'est la représentation correspondant à la soif qui guide notre recherche, et idem pour la faim. Avec ce type de modifications de l'algorithme, on peut conserver les propriétés de convergence pour les deux représentations.

3.3 Intérêt du Q-learning

Nous avons déjà dit que l'intérêt pour le Q-learning s'était ravivé à la suite des liens qui semblaient établis avec un cadre mathématique formel, la programmation dynamique. Notre démonstration de la convergence du Q-learning à un pas avec un seul objectif confirme les relations que l'on pouvait attendre entre Q-learning et programmation dynamique. Cependant cette démonstration met en lumière les limites des rapprochements que l'on peut établir entre DP et Q-learning. Pour les deux algorithmes on cherche à maximiser la somme des récompenses, mais le Q-learning ne peut réussir à construire convenablement des surfaces à plusieurs lieux de récompenses de la même manière que la programmation dynamique.

C'est donc ailleurs qu'il faut chercher un intérêt au Q-learning. Lorsqu'il n'y a qu'une seule récompense, le Q-learning construit une solution optimale dans l'espace exploré. Mais surtout, c'est le mode de construction de la surface par Q-learning qui paraît être général pour l'apprentissage par renforcement. On peut distinguer deux propriétés fondamentales de la construction de cette surface, que l'on peut considérer de façon imagée comme une montagne à un sommet :

- le sommet s'élève du fait d'un ajout à chaque itération dans la case qui constitue l'objectif. (1)
- la montagne s'étend, se propage, à chaque itération une case du chemin emprunté jusqu'à la sortie, valuée jusque là à 0 est modifiée pour prendre une valeur non nulle. (2)

Ces deux dernières propriétés semblent nécessaires pour tout algorithme d'apprentissage par renforcement. Une récompense localisée permet d'obtenir la propriété (1) et va constituer le renforcement. La propriété (2) assure que l'apprentissage est réalisé. En effet il faut pour cela qu'une trace de l'objectif soit accessible en un autre lieu, ce que seul un phénomène de propagation peut assurer.

Le Q-learning apporte en plus des deux propriétés de précédentes, une certaine croissance le long des trajectoires jusqu'à l'objectif à savoir la propriété CVA.

QUATRIEME PARTIE :

AU DELA DU Q-LEARNING

PERSPECTIVES POUR L'APPRENTISSAGE PAR RENFORCEMENT

Introduction

Dans cette quatrième et dernière partie, nous introduisons quatre voies différentes permettant d'étudier plus finement comment opère le Q-learning en tant que méta-dynamique construisant des suites de fonctions de Liapounov. L'objectif est d'établir à partir de ces propriétés de nouveaux algorithmes prolongeant le Q-learning dans le cadre de l'apprentissage par renforcement. Ces algorithmes plus souples doivent pouvoir s'appliquer à d'autres classes de problèmes que celle du labyrinthe avec une unique récompense. Ils doivent permettre un apprentissage, mais aussi une adaptation lorsque l'environnement est changeant.

Nous cherchons à prolonger l'aspect fondamental du Q-learning comme méta-dynamique construisons des fonctions de Liapounov pour plusieurs dynamiques. L'objectif est de conserver des propriétés de convergence ou du moins des comportements mathématiquement compréhensibles pour l'ensemble des qualités pour des problèmes plus larges que ceux posés dans les trois premières parties.

Pour cela dans les paragraphes 4.1 et 4.2, nous nous appuyons sur des modèles provenant de la physique. La caractéristique principale de ces différents modèles est de chercher "un comportement collectif à partir des caractères locaux les plus simples", comme l'explique S. Roux [19]. Ceci correspond à notre hypothèse cognitive de vision ou d'action à une case.

Dans le paragraphe 4.3, nous cherchons à effectuer ces prolongements à partir de modèles et d'idées provenant de la psychologie.

Dans le paragraphe 4.4 enfin, nous cherchons à définir comment l'IA distribuée peut nous aider à modifier notre algorithme, pour éventuellement le comparer à des algorithmes d'évolution comme les algorithmes génétiques.

4.1 Q-learning et phénomènes d'avalanche

Nous avons déjà vu que le but du Q-learning était la construction d'une surface. De plus si l'on ne considère qu'une seule récompense positive, cette surface correspond à une petite montagne régulière ou encore à un tas de sable !!!

a) le modèle de Per Bak

Per Bak a travaillé sur les systèmes critiques auto-organisés et les phénomènes d'avalanche. Par la suite, Stéphane Roux en a fait une analyse en rapprochant cette expérience des phénomènes de percolation. Dans ce cadre, l'exemple paradigmatique est l'expérience du tas de sable que nous allons ici étudier de manière plus approfondie afin d'essayer de faire le lien avec la représentation géométrique du Q-learning.

Nous allons tout d'abord présenter la modélisation due à Per Bak du problème du tas de sable à une dimension. Le modèle a un nombre important d'états métastables augmentant de manière exponentielle avec la longueur du système. De manière étonnante le modèle évolue vers l'état le moins stable de tout ces états métastables qui est appelé l'état stable minimal global.

On a un modèle à une dimension d'une pile de sable de longueur N. Les conditions aux limites sont telles que le sable peut quitter le système uniquement du côté droit. Le nombre z_n représente la différence de hauteur entre deux positions successives le long du tas de sable : $z_n = h(n) - h(n+1)$. La dynamique est la suivante lorsque l'on rajoute un grain de sable en n :

$z_n \rightarrow z_n + 1$

$z_{n-1} \rightarrow z_{n-1} - 1$

et lorsque la différence de hauteur est supérieure à une certaine valeur critique fixée z_c, un grain de sable tombe alors sur le niveau le plus faible:

$z_n \rightarrow z_n - 2$

$z_{n\pm1} \rightarrow z_{n\pm1} + 1$ pour $z_n > z_c$

Les conditions aux limites peuvent être soit ouvertes soit fermées.

En dimension deux l'algorithme se transforme comme suit. La pente z(i,j) en (i,j) est définie comme la différence entre la somme des hauteurs des deux sites amonts et la somme des hauteurs des deux sites avals. Les règles de réarrangement s'écrivent alors :

$z(i\pm1,j) \rightarrow z(i\pm1,j) +1$

$z(i,j) \rightarrow z(i,j) - 4$

$z(i,j\pm1) \rightarrow z(i,j\pm1) + 1$

Il est à noter que dans ce cas, deux réseaux duaux sont superposés, l'un correspondant aux limites entre les grains de sable et l'autre à la hauteur de chaque colonne.

b) intérêt du modèle

en dimension 1 :

Si le système est fermé, il va acquérir un état stationnaire d'équilibre statistique. En particulier, si on part d'une situation plate et si z_c vaut 1, Roux fait remarquer que le modèle devient identique à celui de la structure des chemins minimaux dans un problème de percolation. De plus si la source des particules est toujours localisée au même endroit le problème est dégénéré et va tendre vers un état où toutes les pentes sont critiques.

en dimension 2 et 3 :

L'intérêt principal du modèle de Per Bak est qu'il converge naturellement vers un état statistique critique. On appelle amas l'ensemble des sites qui sont modifiés par l'addition d'une particule en un point donné. On obtient alors à partir de la dimension deux, des amas instables de toutes tailles jusqu'à celle du système lui-même.

Per Bak a essentiellement étudié la statistique de durée T des avalanches pondérée par la réponse moyenne s/T, où s est le nombre de particules mises en jeu pour la même avalanche. Il a observé une loi puissance pour la distribution d(T) : $d(T) \approx T^{-\alpha}$ avec $\alpha = 0.43$ en dimension deux et $\alpha = 0.92$ en dimension trois.

D'autres lois puissances ont été observées pour la taille n(s) des amas. De même si on appelle θ la pente moyenne de l'empilement, le courant $J(\theta)$ de particules, nombre de grains de sable qui sortent du système lorsque l'on rajoute un grain au tas, la longueur de corrélation $\xi(\theta)$, la "susceptibilité" $\chi(\theta)$, le temps de relaxation d'un amas de taille l t(l) ou la dimension fractale D des amas instables obéissent à des lois puissances.

Hwa et Kardar ont même proposé une version continue de ce modèle.

c) un nouvel algorithme

La première idée que donne le modèle de Per Bak est de modifier les qualités autrement. Dans une case, on applique T non plus seulement à la case dans laquelle on vient de passer, mais à l'ensemble des cases voisines dont on peut considérer qu'elles auraient pu être antécédentes avec la propriété CVA. De cette manière la convergence de l'algorithme est conservée, mais en plus la convergence est très fortement accélérée car en une itération, on modifie beaucoup plus de qualités qu'avec les algorithmes précédents. On obtient ainsi une surface proche de celle cherchée en beaucoup moins d'itérations. Les hypothèses cognitives sont conservées car on ne modifie les qualités que de cases que l'on peut percevoir.

d) Apprentissage et avalanches

On peut même essayer d'utiliser directement l'algorithme de Per Bak pour modifier les qualités des cases.

La souris déclenche les lois locales lorsqu'elle arrive sur les cases.
De cette manière, des paramètres comparables à ceux du modèle de Per Bak devraient suivre des lois puissances. Il faudra alors comparer le Q-learning et ce nouvel algorithme assez différent même si tous deux cherchent à atteindre un objectif proche, construire une fonction croissante ou presque croissante le long des trajectoires, c'est à dire des fonctions qui sont pratiquement des fonctions de Liapounov. L'approche de Per Bak est beaucoup plus dynamique et les phénomènes observés sont moins monotones.

De plus les avalanches peuvent correspondre éventuellement à un phénomène d'oubli, elles permettent de ne pas figer la solution obtenue et donc de faire plus d'exploration. Après simulation, il sera intéressant de comparer les résultats numériques obtenus aux expériences sur les rats ce qui pourra nous fournir éventuellement une validation cognitive de ce modèle.

e) Généralisation de cette approche

Tout ce qui précède nous amène à généraliser l'énoncé du problème de départ et nous fournit une méthode de résolution même si certaines étapes sont difficiles à réaliser concrètement.

On se donne un objectif à apprendre. Puis on cherche une dynamique différentiable , en fait une inclusion aux différences pour laquelle l'objectif est point fixe. Parallèlement on doit se donner un espace d'état lié à cette dynamique.

On cherche alors à trouver des opérateurs T' comme T qui construisent des suites de fonctions de Liapounov. Eventuellement ces opérateurs peuvent construire des fonctions "presque" de Liapounov dans un sens que nous n'avons pas formalisé rigoureusement, mais l'algorithme de Per Bak en est semble-t-il l'exemple paradigmatique. L'algorithme de Per Bak nous propose donc un modèle pour modifier la méta-dynamique T en T'. T' ne construisant pas nécessairement des fonctions de Liapounov, mais des fonctions suffisamment proches pour que les dynamiques ensuite effectivement suivies nous mènent presque sûrement à l'objectif.

Nous cherchons en fait à élargir la méta-dynamique proposée par le Q-learning qui nous amène de façon certaine à l'objectif. Nous cherchons une méta-dynamique T' qui nous amène à l'objectif avec une certaine probabilité que nous devons pouvoir régler pour être aussi proche de 1 qu'on le souhaite.

De plus comme l'a fait remarquer Stéphane Roux, on peut alors considérer que l'algorithme agit comme un modèle de percolation pour lequel on peut chercher à établir numériquement les coefficients des lois puissances sous-jacentes.

4.2 Q-learning et problèmes de connexité

a) introduction

Nous avons déjà vu que la convergence de l'algorithme était liée à une certaine forme de connexité des case-actions qui sont choisies et modifiées dans l'espace d'état quand on se déplace dans l'espace physique. C'est cette connexité que nous avons réussi à retrouver en établissant un nouveau Q-learning à deux cases (voir paragraphe 2.5).

Une autre forme de connexité apparaît et nous semble intéressante, elle correspond à la localisation et à la forme des raccords des cases qui étaient valuées à zéro et sont désormais valuées à une valeur non nulle.
Dans ce cadre, le Q-learning semble agir comme le modèle DLA que nous rappelons brièvement ici

b) le modèle DLA [19]

Le modèle d'Agrégation Limitée par Diffusion (DLA) peut être présenté comme un modèle de diffusion. On se donne un ensemble connexe de sites (amas). Une particule, lâchée à l'infini, diffuse librement jusqu'à venir sur l'un des sites. Ce site est alors incorporé à l'amas. Une nouvelle particule est de nouveau lâchée de l'infini. Dès qu'elle arrive en contact avec le nouvel amas, le site sur lequel elle se trouve est ajouté à l'amas, et ainsi de suite, le processus est itéré. Dans notre problème, nous sommes intéressés par une géométrie pour laquelle l'amas de départ est un site unique, le lieu où la souris est récompensée. Le problème peut également être présenté en termes de potentiel V(x), qui doit satisfaire l'équation de Laplace

$$\Delta V = 0$$

avec les conditions aux limites suivantes :
- sur l'amas $V(x) = 0$
- à l'infini $V(x) = 1$

La difficulté d'étude théorique de ce type de problème dans la limite continue provient des conditions aux limites qui sont variables dans le temps (problème d'évolution à frontière libre)

La structure de l'amas possède alors plusieurs propriétés caractéristiques. Tout d'abord l'instabilité de la géométrie de l'amas donne lieu à une structure arborescente ayant pour racine l'amas initial.

Plusieurs simulations sont présentées par Roux [19]. Des lois puissances peuvent être mises en évidence mesurant la croissance de l'amas selon différentes directions. Leurs exposants critiques sont variables selon la géométrie du réseau formé par les sites. Dans le cas du DLA sur réseau Euclidien, qui correspond aux discrétisations actuellement pratiquées quand on utilise l'algorithme du Q-learning, la distribution de probabilité de croissance sur la frontière de l'amas est multifractal.

c) test du modèle

Pour pouvoir établir des comparaisons, à partir des simulations, entre la façon dont œuvre le Q-learning et le modèle DLA, nous proposons des modifications dans l'implémentation du Q-learning. Tout d'abord, si l'on veut simuler le fait que l'on envoie les particules de l'infini, je propose que la sortie soit au centre du labyrinthe, et que le labyrinthe soit carré. De plus les souris au lieu d'avoir pour point de départ une case quelconque du labyrinthe, démarreront d'une case quelconque sur les bords du labyrinthe. Ensuite, si l'on veut simuler un modèle DLA isotrope, il faut construire un labyrinthe sans obstacle, ou possédant une répartition d'obstacles symétrique.

Cette implémentation ne peut être intéressante que sur de grands labyrinthes, pour lesquels le temps nécessaire pour construire une bonne représentation pour la souris risque d'être long, la recherche s'effectuant au départ au hasard.

L'intérêt de ce type de simulation ou de comparaisons avec des modèles physiques est de chercher au moins en probabilité quelles lois régissent le passage d'une qualité nulle à une qualité non nulle. En reprenant les notations du paragraphe 2.2, nous cherchons à établir ici la manière dont la suite C_n croît. Cette étude est essentielle, car elle permettra de comprendre les phénomènes liés à la phase exploratoire pendant laquelle un nombre important de case-actions possède des qualités nulles. De cette manière, nous pourrons estimer la durée de la phase exploratoire et éventuellement la réduire. En particulier, la connaissance des lois qui régissent en probabilité la croissance de l'amas selon la géométrie du réseau peut nous permettre de définir selon notre problème le type de discrétisation, c'est à dire le type de pavage de notre espace d'état que nous avons intérêt à choisir

4.3 Les signes en chemin

Souvent dans le monde réel, on peut être guidé grâce à des signes intermédiaires [8]. On considère que ces signes ont été évalués, et on les utilisera ici directement sous forme de valeurs numériques. Ils n'ont pas ici le même statut que les récompenses correspondant à un objectif et ils ne soulèvent pas les mêmes problèmes que ceux

rencontrés pour le Q-learning à plusieurs récompenses. Nous avons défini un algorithme qui permet d'en tenir compte tout en restant convergeant avec la même démonstration que celle du Q-learning à un pas.

Les signes rencontrés, positifs ou négatifs ne peuvent pas modifier directement la représentation que l'on se fait de l'espace d'état, sinon on ne conserve pas la propriété CVA sauf dans quelques cas particuliers. Par contre ces signes sont pris en compte dans le choix de la case suivante. On adapte notre algorithme Q3 en appelant s(a) l'estimation du signe au point a. On définit alors l'algorithme Q4 comme suit :

algorithme (Q4) :

Soit B l'ensemble des successeurs de a qui vérifient la propriété CVA
B' ensemble des successeurs b de a qui vérifient Q(b) = 0
P une procédure de tirage selon une distribution de Boltzmann

-si Q(a)≠0 choix(Suc(a)) = P(Q(b) + s(b)) avec b appartient à B
-si Q(a)=0,

- trois fois sur quatre, choix(Suc(a)) = P(Q(b) + s(b)) avec b appartient à B'
- une fois sur quatre, choix(Suc(a)) = P(Q(b) + s(b)) avec b appartient à B

(Q4)

4.4 Le Q-learning multi-souris

Nous avons défini un nouvel algorithme utilisant le Q-learning avec plusieurs souris. Nous avons défini ce nouvel algorithme pour obtenir par l'intermédiaire du groupe, une probabilité plus grande que les représentations des souris correspondent à la solution optimale du problème que donnerait un observateur extérieur.

Plusieurs souris sont placées dans un labyrinthe, chacune possédant et construisant sa propre représentation grâce à l'algorithme Q2 ou Q3. Chaque souris effectue ses actions simultanément, ce que l'on simule en faisant agir chaque souris l'une après l'autre.

La nouveauté de l'algorithme se situe lorsque deux souris 1 et 2 se trouvent au même moment dans la même case. Dans ce cas ces souris vont échanger de l'information. La propriété CVA définit pour chaque souris selon sa représentation un sous ensemble de successeurs acceptables puisque les choix se font selon les algorithmes Q2 et Q3 ou Q4. Soit S1, l'ensemble des successeurs acceptables de la souris 1 et S2 celui de la souris 2.

Si $S1 \cap S2 = \emptyset$ alors la souris 1 et la souris 2 effectuent leur choix comme d'habitude.

Si $S1 \cap S2 \neq \emptyset$ alors la souris 1 choisit comme action celle qui est optimale pour la souris 2 et la souris 2 choisit comme action celle qui est optimale pour la souris 1. Eventuellement, on effectue un tirage au sort si plusieurs actions vérifient les critères précédents.

Dans cet algorithme, différentes hypothèses sont faites sur nos souris artificielles. Tout d'abord, nous conservons notre hypothèse de vision locale. Une autre hypothèse cognitive déjà utilisée dans tout le rapport apparaît plus clairement : chaque souris peur seulement accéder au résultat d'une opération sur les qualités, par exemple le choix d'une action, mais elle n'a jamais accès à la valeur numérique des qualités. Une nouvelle hypothèse cognitive est faite sur les capacités de communication des souris. Les souris sont capables de s'échanger des ensembles. Elles sont par contre incapables de communiquer les valeurs des qualités de leurs représentations car elles n'ont pas accès directement à ces qualités.

Ce nouvel algorithme reste convergeant toujours avec le même principe de démonstration. En effet les modifications ne concernent que le choix d'une action parmi un sous-ensemble dont chaque élément convient pour conserver la convergence de l'algorithme. L'algorithme continue à construire des fonctions de Liapounov pour chaque souris, et pour chacune selon sa dynamique.

L'intérêt de cet algorithme est multiple. Le premier est que l'on peut espérer que certaines souris trouveront une solution qu'un observateur extérieur considèrerait comme optimale. On peut espérer qu'en probabilité, on obtiendra la solution optimale plus fréquemment. Le second intérêt est éventuellement une meilleure adaptabilité en utilisant l'algorithme définit avec PROC1, PROC2 et PROC3. Enfin si une nouvelle souris possédant une représentation vierge est introduite dans le labyrinthe, son apprentissage sera beaucoup plus rapide, car pendant toute la période qui devrait être de l'exploration, elle profitera de l'expérience des autres souris. De cette manière, on peut attribuer une durée de vie aux souris et en faire naître de nouvelles possédant une représentation initialisée à zéro.

En établissant un algorithme avec une population de souris, nous pourrons définir des tests pour comparer les capacités d'adaption lorsque la population a été modifiée par apprentissage ou par évolution. En effet, en collaboration avec Laurent Atlan, nous voulons comparer sur les mêmes problèmes, une population qui a suivi une procédure de Q-learning, et une autre modifiée par algorithme génétique.

CONCLUSION

Dans notre travail, les aspects cognitifs sont importants. Nous avons toujours travaillé en supposant que la vision n'avait lieu qu'à une case. Cependant cette hypothèse cognitive n'est pas trop restrictive puisque cette case peut être aussi grande qu'on le souhaite. Elle peut correspondre à notre horizon visuel effectif. De même le mot vision n'est pas tout à fait adéquat. Le mot perception correspondrait mieux puisque nous travaillons dans un espace d'état de dimension quelconque dans lequel chaque dimension peut correspondre à un paramètre perceptif comme la couleur ou l'odeur.

Les expériences d'étude du comportement des rats [22], [23], [24], [25] peuvent nous permettre de vérifier que les mécanismes en jeu dans les algorithmes d'apprentissage du même type que le Q-learning sont cognitivement plausibles. Ces expériences peuvent également nous aider à définir de nouveaux algorithmes.

L'apport essentiel de notre travail est certainement la description du Q-learning comme méta-dynamique générant des suites de fonctions γ-Liapounov. Ce résultat est intéressant car à notre connaissance aucune étude mathématique de ce type d'opérateur n'a été réalisée jusqu'à présent.

Dans notre quatrième partie, nous avons commencé à analyser le Q-learning à partir de modèles physiques n'effectuant que des modifications locales. Le modèle de Per Bak peut nous permettre de modifier l'opérateur T de la méta-dynamique, alors que le modèle DLA nous permet d'analyser comment s'effectue la croissance du domaine C_n pour lequel les qualités des case-actions sont non nulles.

De même une analyse de la méta-dynamique peut se faire en terme d'inclusions différentielles. Aussi le formalisme de la théorie de la viabilité [20], [21], peut nous aider à reformuler les propriétés de l'algorithme du Q-learning pour définir ensuite de nouveaux algorithmes d'apprentissage.

BIBLIOGRAPHIE :

Sur le Q-learning :

[1] A.G. Barto, R.S. Sutton & C.W. Anderson, (1983). *Neuronlike adaptative elements that can solve difficult learning control problems*, IEEE, vol smc-13 (5) : 834-846.

[2] A.G. Barto, S.J. Bradtke, S.P. Singh (1991). *Real-time learning and control using asynchronous dynamic programming*, Technical report 91-57, August 1991. University of Massachussetts.

[3] P. Dayan, *Navigating through temporal difference.*

[4] S.B. Thrun, K. Möller, A. Linden, *Planning with an adaptative world model.*

[5] R.S. Sutton, *Integrated modeling and control based on reinforcement learning and dynamic programming.*

[6] R.S. Sutton, (1990). *Planning by incremental dynamic programming.*

[7] R.S. Sutton, (1990). *Integrated architectures for learning, planning, reacting based on approximating dynamic programming.* Proceedings of the Seventh International Conference on Machine Learning.

[8] L.J. Lin, *Self-improvement based on reinforcement learning, planning and teaching.*

[9] R. Munos, (1992). *Algorithmes génétiques et apprentissage par renforcement.* Rapport de DEA de Sciences Cognitives, EHESS, Paris.

[10] S.P. Singh, *Transfer of learning across compositions of sequential tasks.*

[11] L.E. Wixson, *Scaling reinforcement learning techniques ,via modularity.*

[12] C. Watkins (1989). *Learning from delayed rewards.* PhD Thesis, University of Cambridge, England.

[13] C. Watkins, P. Dayan (1992).*Technical Note : Q-learning.* Machine Learning, 8, 279-292.

Sur la programmation dynamique :

[14] S. Ross, (1983). *Introduction to stochastic dynamic programming.* Academic Press, New York.

[15] M. Gondran, M. Minoux, (1979). *Graphes et algorithmes.* Eyrolles .

[16] M.P. Bertsekas, J.N. Tsitsiklis, (1989). *Parallel and distributed computation, numerical methods.* Prentice-Hall.

Sur les systèmes dynamiques :

[17] R.L. Devaney, (1989). *An introduction to chaotic dynamical systems.* Addison Wesley.

Sur la percolation et les systèmes critiques auto-organisés :

[18] Per Bak, C. Tang, K. Wiesenfeld, (1988). *Self-organised criticality*, Physical Review A vol 38, n°1.

[19] S. Roux, (1990). *Structures et désordres*, thèse d'état, Ecole Nationale des Ponts et Chaussées, Paris.

Sur le contrôle :

[20] N. Seube, (1992). *Régulation de systèmes contrôlés avec contraines sur l'état par réseaux de neurones,* thèse, Université Paris IX Dauphine, Paris.

[21] J.P. Aubin, (1991). *Viability theory,* Birkhauser.

Résultats d'expériences psychologiques sur les rats :

[22] M. Blancheteau, *Motivations et guides des déplacements chez l'animal*, thèse d'état, Paris.

[23] M. Blancheteau, *Apprentissage chez l'animal.*

[24] M. Blancheteau, *Influence des attitudes sur la perception visuelle.*

[25] M. Blancheteau, *L'orientation spatiale chez l'animal, indices et repères.*

[26] R.A. Rescorla & A.R. Wagner, (1972). *A theory of Pavlovian conditioning : variations in the effectiveness of reinforcement and nonreinforcement.* In Black & Prokasy (Ed.), Classical conditioning II. Appleton-Century-Crofts.

ANNEXE 1

Définition 1 :

Soit A un sous-espace fini de Z^p et $a \in A$, a de coordonnées $a_1, \ldots, a_p$.

On appelle c **case** de A le sous-espace de R^p défini par : $\exists\, a \in A /$

$$a_1 \leq x_1 \leq a_1 + 1$$
$$\ldots$$
$$a_p \leq x_p \leq a_p + 1$$

et l'ensemble des p-uplets $(a_1 + \varepsilon, \ldots, a_p + \varepsilon) \in A$ avec $\varepsilon \in \{0 ; 1\}$

On particularise le p-ième axe qui correspondra à l'espace des actions de dimension 1.
La p-ième coordonnée x_p correspond donc à une action. On parle de case-action, les p-1 premières coordonnées se situant dans un espace d'état de dimension p-1 et la dernière dans l'espace des actions.

Il est bon de choisir A tel que ses éléments définissent des cases
Cela marche très bien si A est la discrétisation d'un espace.
Tout ceci reste valable si on se donne seulement un espace de coordonnées, les cases ne seront alors simplement ni des carrés, ni des rectangles, mais peuvent avoir une certaine courbure.

On appelle C l'ensemble des cases de A.

Définition 2 :

On appelle **successeur possible de c $\in$ C**, l'ensemble des éléments de C qui possèdent 2^{p-1} points de A communs avec la projection de c sur les p-1 premiers axes, si l'on suppose que les actions opèrent un déplacement d'une case, dans une des cases adjacentes, et l'on prend alors comme successeur toutes les case-actions de ces cases adjacentes. C'est bien le cas dans l'expérience du labyrinthe. Sinon on définit le successeur de c comme l'ensemble des éléments de C dont la projection sur les p-1 premières coordonnées peut être accessible par une des actions possibles en c. Plus concrètement, quant l'effet des actions possibles est le même en toute case et qu'il est connu, la première définition suffit. Sinon la seconde est nécessaire.

On définit **Suc : C -> C la fonction multivoque** qui à $c \in C$ associe $c' \in C$ et c' un successeur de c. Nous écrirons généralement c' = Suc {c}, les accolades notant la multivocité. Nous notons aussi $c' \in \{Suc(c)\}$, lorsque nous considérons l'ensemble des successeurs de c.

Définition 3 :
Soit Q : C -> [0 ; 1];
On appelle **surface associée à Q et C** le sous ensemble S de dimension p de R^pxR / $\{s \in S$ ssi $\exists\, c \in C / s = (c, Q(c))\}$. On dira simplement surface par la suite.

Définition 4 :
On appelle **surface à un seul objectif** une surface pour laquelle il existe un et un seul $s = (c, Q(c)) \in S$ tel que $0 < Q(c)$ et $Q(c) = \max_{c' \in C} \{ Q(c') \}$. On appellera C_f le point de R^p associé.

Définition 5 : une surface à un objectif est **bien stratifiée selon (P)** ssi $\forall$ c $\in$ C $\exists$ p $\in$ N, $\exists$ $(c_n)_{0\leq n\leq p} \in C$ /{ $c_0 = c$; $c_p = C_f$; $\forall$ n / $1\leq n\leq p$, $c_n = Suc\{ c_{n-1}\}$} (1)
la suite (c_n) vérifie (P) (2)
On appellera **chaîne partant de c et longueur p**, U(c) une telle suite $(c_n)_{0\leq n\leq p}$ d'éléments de C

(P) est l'une des deux propositions suivantes :

(P1) Soit une suite $(c_n)_{0\leq n\leq p} \in C$ et γ fixé, $\gamma \in [0;1]$;

$\forall$ n / $1 \leq n \leq p$, $Q(c_n) \geq Q(c_{n-1})/\gamma$ (3)

$\forall$ n / $1 \leq n \leq p$, $Q(c_n) = \max Q(Suc\{c_{n-1}\})$ (4)

(P2) Comme (P1), mais sans la propriété (4)

Nous nous intéresserons plus particulièrement à (P2) qui est une extension de (P1), ne se limitant plus au choix de la qualité maximale.
Nous définissons alors la propriété de croissance de la valeur actualisée (CVA) :

propriété CVA : soit γ un nombre entre 0 et 1, soient c et c' deux case-actions, c' étant un successeur possible de c. Soit Q la fonction qui à une case-action associe sa qualité,

c et c' vérifient la propriété CVA ssi $Q(c) \leq \gamma Q(c')$

f : C->C vérifie la propriété CVA en c ssi c et f(c) vérifient la propriété CVA

f est CVA par rapport à Q ssi f vérifie la propriété CVA en tout point c $\in$ C

Par la suite nous dirons CVA, sans préciser que c'est par rapport à Q.

Dans ce cadre, on peut donner une autre définition de la propriété "bien stratifiée" :

définition : On dira que la fonction Q de C dans C qui à chaque case-action associe sa qualité est bien stratifiée ssi

$\forall$ c $\in$ C, { c' $\in$ {Suc(c)} / $Q(c) \leq \gamma Q(c')$ } $\neq \emptyset$

On définit un **opérateur T : [0; 1] -> R**
Soit $\alpha \in [0; 1[$.
L'opérateur T s'applique sur un élément c de C de la manière suivante :

si $c \neq C_f$, $T(Q(c)) = Q(c) + \alpha(\gamma Q(Suc\{c\}) - Q(c))$

et $T(Q(C_f)) = Q(C_f)(1 - \alpha) + \alpha$

On dit qu'un opérateur T opère sur une chaîne U(c) ssi T opère successivement sur chacun des éléments $c_0, c_1, ..., c_p$ de la chaîne.

Proposition 1 : T a une image dans [0; 1] :

démonstration :

$T(Q(C_f)) = Q(C_f)(1-\alpha)+\alpha$
$\leq 1-\alpha+\alpha$
(car $0 \leq Q(C_f) \leq 1$ et $1-\alpha \geq 0$)
≤ 1

$T(Q(c)) = Q(c)(1-\alpha)+\alpha\gamma\, Q(Suc\{c\})$
$\leq Q(C_f)(1-\alpha)+\alpha\gamma\, Q(C_f)$
$\leq Q(C_f)(1-\alpha+\alpha\gamma)$
$\leq 1+\alpha(\gamma-1)$
≤ 1 (car $0 \leq \gamma \leq 1$)

$T(Q(C_f)) \geq \alpha \geq 0$

$T(Q(c)) = Q(c)(1-\alpha)+\alpha\gamma\, Q(Suc\{c\})$
or $\alpha\gamma \geq 0$;
$1-\alpha \geq 0$;
$Q(c) \geq 0$;
$Q(Suc\{c\}) \geq 0$;
donc $T(Q(c)) \geq 0$

Proposition 2 : Soit S une surface associée à Q et C. Si S est une surface à un objectif bien stratifiée selon (P) alors $\forall\, c \in C$ et une chaîne U(c), si T opère sur U(c) la nouvelle surface S' après opération est également à un objectif et bien stratifiée selon (P).

démonstration :

(i) Puisque T a une image dans [0;1], chaque élément s' de S' est bien un couple dont le premier élément appartient à C et le second une variable à valeur dans [0; 1] .

(ii) S' est à un objectif qui est celui de S :

- $T(Q(C_f)) - Q(C_f) = \alpha(1 - Q(C_f))$
≥ 0 (car $\alpha \geq 0$ et $Q(C_f) \leq 1$)
- les seuls points modifiés par T sont ceux de la chaîne U(c) donc pour les autres points $d \in C$, on a toujours $T(Q(C_f)) \geq Q(C_f) > Q(d) = T(Q(d)) \geq 0$

- pour les points de la chaîne différents de l'objectif :

si $d \in U(c)$, on a

$T(Q(d)) = Q(d)(1-\alpha)+\alpha\gamma\, Q(Suc(d))$
$< Q(C_f)(1-\alpha)+\alpha\gamma\, Q(C_f)$
$= Q(C_f)(1-\alpha+\alpha\gamma)$

$= Q(C_f)\ (1 + \alpha(\gamma - 1))$
$\leq Q(C_f)$

car $-1 < \gamma - 1 \leq 0$

soit $-\alpha < \alpha\ (\gamma - 1) \leq 0$

soit $0 < 1 - \alpha < 1 + \alpha\ (\gamma - 1) \leq 1$

Donc $T\ (\ Q(d)\ ) < Q(C_f) \leq T\ (\ Q(C_f)\ )$

- Finalement $\forall\ d \in C,\ d \neq C_f$, on a $T\ (\ Q(d)\ ) < T\ (\ Q(C_f)\ )$

Donc S' est à un objectif et c'est celui de S.

(iii) S' est bien stratifiée selon (P).

Il suffit de montrer que pour tout point $d \in C$, on peut construire une suite finie de C qui vérifie (P)

a) si $d \in U(c)$, $\exists\ k \leq p$ tel que $d = c_k$,

alors la chaîne $c_k, ..., c_p$ convient :
il est clair que cette suite vérifie toujours (1) car le nouvel objectif est aussi l'ancien objectif et que les anciens successeurs sont encore des successeurs (cette propriété valable dans le cas déterministe, ne l'est plus en général si on a une probabilité de transition)
Montrons que cette suite vérifie (P) :
T a été appliquée à la chaîne $(c_k, ..., c_p = C_f)$
et d'après la propriété CVA de la chaîne avant l'application de T

$Q(c_k) \leq \gamma\ Q(c_{k+1})$

...

$Q(c_{p-1}) \leq \gamma\ Q(c_p)$

$\gamma\ T\ (\ Q(c_p)\ ) - T\ (\ Q(c_{p-1})\ )$

$= \gamma\ T\ (\ Q(C_f)\ ) - Q(c_{p-1})\ (1 - \alpha) - \alpha\gamma\ Q(C_f)$

$\geq \gamma\ Q(C_f) - Q(c_{p-1})\ (1 - \alpha) - \alpha\gamma\ Q(C_f)$

{on a déjà montré que $T\ (Q(C_f)) \geq Q(C_f)$ }

$\geq (1 - \alpha)\ (\gamma\ Q(C_f) - Q(c_{p-1}))$

≥ 0 { d'après la propriété CVA }

$\forall$ m tel que $k + 1 \leq m \leq p - 1$

$\gamma\ T\ (\ Q(c_m)\ ) - T\ (\ Q(c_{m-1})\ )$

$= \gamma\ Q(c_m) + \alpha\gamma\ (\gamma\ Q(c_{m+1}) - Q(c_m))$

$- Q(c_{m-1}) - \alpha\ (\gamma\ Q(c_m) - Q(c_{m-1}))$

$= \gamma\ Q(c_m)\ (1 - \alpha) + \alpha\gamma\ (\gamma\ Q(c_{m+1}) - Q(c_m))$

$+ Q(c_{m-1})\ (-1 + \alpha)$

$= (1 - \alpha)\ (\gamma\ Q(c_m) - Q(c_{m-1}))$

$+ \alpha\gamma\ (\gamma\ Q(c_{m+1}) - Q(c_m))$

≥ 0 { car $\gamma\ Q(c_m) \geq Q(c_{m-1})$ et $\gamma\ Q(c_{m+1}) \geq Q(c_m)$ }

donc on a la chaîne précédente issue de d qui vérifie bien la propriété CVA pour S', c'est à dire la fonction T(Q)

b) si $d \notin U(c)$:

comme S était bien stratifiée selon (P), $\exists\ d_0, ..., d_p$ pour S tels que
$d_p = C_f$
...
$d_k = Suc\ \{d_{k-1}\}$ pour $1 \leq k \leq p$
$d_0 = d$

α) si $\forall\ k\ /\ 1 \leq k \leq p$ on a $d_k \notin U(c)$
alors aucun élément de la chaîne $d_0, ..., d_p$ n'a été modifié et cette chaîne vérifie toujours (P) pour S' donc il convient à partir de d pour S'.

β) si $\exists\ k\ /\ 1 \leq k \leq p$ et $d_k \in U(c)$
Prenons le k minimum qui vérifie $d_k \in U(c)$, ce minimum existe car le sous-ensemble [1 .. p] de N est fini.
Donc $\exists\ m\ /\ d_k = c_m$
Montrons alors que la chaîne
$d_0, d_1, ..., d_{k-1}, d_k = c_m, c_{m+1}, ..., C_f$ convient :
- cette suite vérifie évidemment (1)
- puisque la suite (d_n) vérifiait (P) pour S, on a

$Q(d_0) \leq \gamma\ Q(d_1)$
$Q(d_1) \leq \gamma\ Q(d_2)$
...
$Q(d_{k-1}) \leq \gamma\ Q(d_k) = \gamma\ Q(c_m)$
ces inégalités sont encore valables après opération de T car T ne modifie pas ces valeurs :
$T(Q(d_0)) \leq \gamma\ T(Q(d_1))$
$T(Q(d_1)) \leq \gamma\ T(Q(d_2))$
...
$T(Q(d_{k-1})) \leq \gamma\ Q(d_k)$

Dans le a), on a montré que puisque $c_m \in U(c)$ on a :
$T(Q(c_m)) \leq \gamma\ T(Q(c_{m+1}))$
...
$T(Q(c_{p-1})) \leq \gamma\ T(Q(c_p)) = \gamma\ T(Q(C_f))$

De plus $T(Q(d_k)) = T(Q(c_m))$
$= Q(c_m) + \alpha\ (\gamma\ Q(c_{m+1}) - Q(c_m))$
$\geq Q(c_m)$

Donc $T(Q(d_{k-1})) \leq \gamma\ Q(d_k) \leq \gamma\ T(Q(d_k))$
et le raccord est effectué, finalement la chaîne précédemment construite vérifie la propriété CVA pour S'

Conclusion : S' est bien stratifiée selon (P) et S' est à un objectif.

Proposition 3 : Si T opère, alors T opère en C_f et la suite des opérés par T de $Q(C_f)$ converge vers 1

démonstration :

on a vu que $\forall\, c \in C$ et si T opère sur c alors il existe une chaîne U(c) telle que T opère sur cette chaîne et donc T opère sur C_f, dernier élément de cette chaîne.

Posons $x_0 = Q(C_f)$
$x_1 = T(x_0)$
...
$x_{k+1} = T(x_k)$

Alors :

$x_1 = x_0 (1 - \alpha) + \alpha$

$x_1 - x_0 = \alpha (1 - x_0)$

- Si $x_0 = 1$ alors $x_1 = 1$ et la suite est stationnaire égale à 1
- Si $0 \leq x_0 < 1$

$x_1 - x_0 = \alpha (1 - x_0) > 0$
...
$x_{n+1} - x_n = \alpha (1 - x_n) > 0$

Donc la suite $x_n = T^n(x_0)$ est strictement croissante et majorée par 1
{ déjà vu, car T : [0 ; 1] -> [0 ; 1] }

La suite (x_n) est croissante et majorée donc convergente.
Soit l la limite de (x_n) alors puisque T est continue pour le point C_f, on a :

$l = l (1 - \alpha) + \alpha$

$l (1 - 1 + \alpha) = \alpha$

$l \alpha = \alpha$

$l = 1$

La suite des itérés de $Q(C_f)$ est croissante et converge vers 1.

Proposition 4 : la suite des itérés en un point $c \in C$ est convergente :

démonstration :

$\forall\, c \in C, \exists\, U(c) = \{ c_0 ; ... ; c_p \}$

$T(Q(c)) - Q(c) = T(Q(c_0)) - Q(c_0)$
$= Q(c_0) + \alpha (\gamma Q(c_1) - Q(c_0)) - Q(c_0)$
$= \alpha (\gamma Q(c_1) - Q(c_0))$
≥ 0 (car on a la propriété CVA sur la chaîne U(c))

Donc la suite des itérés par T en un point $c \in C$ de Q(c) est croissante. De plus elle est majorée par 1 donc elle est convergente.

Conclusion : $\forall\, c \in C$ la suite des itérés par T de Q(c) est convergente.

Proposition 5 : pour les chaînes dont les éléments sont valués strictement positifs, la solution est sans boucle et les chaînes sont des chemins.

démonstration :
Si dans une chaîne issue de c on a un k minimal tel que $Q(c_k) > 0$
alors $Q(c_k) \leq \gamma\, Q(c_{k+1})$
donc tous les points suivants c_{k+1}, ..., c_p sont tels que
$Q(c_{k+1}) > 0$
...
$Q(c_p) > 0$
on obtient alors $Q(c_k) \leq \gamma\, Q(c_{k+1}) < Q(c_{k+1})$
Finalement $\forall$ p, q avec $p > k$ et $q > k$, $p \neq q$
on a $Q(c_p) \neq Q(c_q)$
donc $c_p \neq c_q$
Ceci démontre bien que notre chaîne est sans boucle.
Du fait de la propriété CVA les successeurs ont des valeurs croissantes le long d'une chaîne. Comme la chaîne est sans boucle, la croissance est stricte et l'on obtient des chemins.

Proposition 6 : dans chaque case c la limite est $1/\gamma^p$ avec p longueur minimale des chemins partant de c. Il s'agit donc de la solution de la programmation dynamique.

Démonstration :
Supposons que chaque case $c \in C$ est telle que la suite des itérés par T de Q(c) converge vers lc

Alors si $c_k \neq C_f$
$T(Q(c_k)) = Q(c_k) + \alpha(\gamma\, Q(Suc(c_k)) - Q(c_k))$
Le problème est que $Suc(c_k)$ est un point différent à chaque itération.

Soit $M \in N$, puisque l'espace d'état est fini, il existe un nombre fini de chemins or il est possible d'effectuer un nombre d'itérations aussi grand que l'on veut car d'après la démonstration précédente, on est sûr d'effectuer chaque itération en un nombre fini de coups. Donc à partir d'un nombre de coups suffisants, on est sûr que l'on passera plus de M fois par l'un des successeurs de c_k, que l'on appellera c'_k. Donc il existe une sous suite des itérés de $Q(c_k)$ qui prend toujours c'_k comme successeur de c_k et qui possède plus de M termes, M étant aussi grand que l'on veut. Pour cette sous-suite, on a univocité du successeur de c_k et donc on peut passer à la limite dans l'équation. On appelle l'c_k, la limite de la suite extraite des itérés en c_k avec pour successeur c'_k. On désigne par lc_k, la limite de la suite en c_k.

$T(Q(c_k)) = Q(c_k) + \alpha(\gamma\, Q(Suc(c_k)) - Q(c_k))$
soit $l'c_k = l'c_k + \alpha(\gamma\, lc'_k - l'c_k)$

alors $l'c_k = \gamma\, lc'_k$
de plus, on sait que la suite des itérés de $Q(c_k)$ est convergente donc elle admet une unique limite qui est aussi celle de la sous-suite extraite précédente donc $lc_k = l'c_k$
Finalement, $lc_k = \gamma\, lc'_k$
Par itération, on obtient puisque la limite en C_f est 1,
$lc_k = \gamma^n$, avec n longueur unique des arcs parcourus plus de M_0 fois avec M_0 entier plus grand que M.

Nous avons démontré jusqu'à présent que si l'on enclenchait convenablement le mécanisme, le Q-learning convergeait. Il reste à démontrer que tout se passe bien au début, ce que nous vérifions avec la proposition 7 :

Proposition 7 : la surface associée à Q_0, telle que toutes les qualités sont nulles est bien stratifiée quel que soit le choix des successeurs.

Démonstration :

Supposons que chaque case $c \in C$ a une qualité nulle, alors quel que soit $c' \in C$, successeur de c, on a $Q_0(c) \leq \gamma\, Q_0(c')$.

En effet tout point $c \in C$ est tel que $Q_0(c) = 0$, et il en est de même pour les points c' donc on a $Q_0(c) = \gamma\, Q_0(c') = 0$ et la proposition 7 est bien vérifiée.

Si on choisit deux points quelconques de C, la propriété CVA est bien vérifiée pour ces deux points et la fonction Q_0, ce qui nous assure que le processus fonctionne bien au départ.