École Nationale Supérieure d'Informatique
et d'Analyse des Systèmes

Mémoire de Projet de Fin de 2ème année

Filière : Ingénierie e-Logistique

Sous le thème:

Amélioration De La Supply Chain Des Dossiers De Retraite En Appliquant L'Algorithme Q-Learning

Présenté et soutenu par :

ZINEB ETTAHAIRY
MOHAMED JARNI

Encadré par :

Mr. ABDELLATIF EL AFIA
MME.HOUDA MEZOUAR

Année Universitaire 2019/2020

Remerciements

Ce travail reflète le fruit des efforts conjugués de plusieurs personnes. Il nous est alors primordial de souligner notre gratitude et notre reconnaissance auprès de toute personne, dont l'intervention au cours de ce projet a favorisé son aboutissement. Même si cette gratitude n'est nullement formulable avec des mots simples.

Tous nos remerciements à notre professeur encadrant Monsieur ***ABDELLATIF EL AFIA*** *qui a accompagné l'élaboration de cet humble projet du début jusqu'à la fin, ainsi que pour ses efforts d'orientation et ses conseils pédagogiques.*

Nous tenons à exprimer nos sincères remerciements à ***MME.HOUDA MEZOUAR,*** *pour tout le temps qu'il nous a consacré, pour son encadrement****,*** *sa confiance, et ses conseils déterminants à la réalisation de ce projet.*

Nos vifs remerciements sont adressés aussi à tous le cadre professoral de l'ENSIAS qui nous assure une formation qualifiante sur le plan technique. Enfin nous remercie toutes les personnes qui ont contribué de près ou de loin à l'élaboration de ce travail. Merci à tous…

Résumé

Le présent rapport constitue le fruit de notre travail accompli dans le cadre de notre projet de fin de 2ème année au sein de l'École Nationale Supérieure d'Informatique et d'Analyse des Systèmes. L'objectif de notre projet était d'appliquer l'algorithme Q_learning au processus métier particulièrement au dossier de retraite.

Pour ce faire, nous avons identifié la problématique concernant le processus opérationnel étudié, qui est en l'occurrence "la gestion des droits à pension civils" du SC de retraite auto-adaptative présenté dans MPF (la caisse de retraite marocaine). Puis, nous avons établir l'évolution de l'algorithme du Q_learning et sa relation avec le processus de décision de Markov ainsi nous avons déterminé ses variances afin de justifier le choix de notre modèle. Après, nous avons passé à l'exploration de la chaine logistique de retraite par les outils de modélisation **SCOR** et **BPMN** en utilisant la modèle DMAIC de Six Sigma pour faire l'évaluation de notre Processus.

C'est donc dans cette optique que notre projet vise à adopter l'algorithme Q_learning pour faire une amélioration de notre système et rendre auto adaptative de chaque état du dossier de retraite.

Table des matières

Listes des figures

Liste des Abréviations

RL : Apprentissage par renforcement

TD : Différence temporelle

BPMN: Business Process Model and Notation

SCOR: Supply Chain Operations Reference

Introduction générale

Parmi les défis que doit relever le régime de retraite, il y a d'une part la nécessité de faire durer ce système plus longtemps. Dans ce cadre, une réforme a été adoptée au Maroc à la fin de 2016, qui ne concerne que le régime civil de retraite. Cette réforme a principalement touché trois points : la cotisation au régime, l'âge de la retraite et le calcul de la pension. Une autre qui visera à une fusion entre les différents fonds peut la suivre. L'autre grand défi est la difficulté de gérer la continuité entre le salaire et la pension de retraite, c'est-à-dire la difficulté d'offrir ce service (la pension de retraite) au client (le retraité) au bon moment (premier mois de sa retraite).

Dans ce travail, ce défi est abordé du côté de la chaîne d'approvisionnement des services, comme exemple des problèmes de continuité, et qui nous fournissent une analyse globale de ce système, et par conséquent nous permettent de mettre en place des stratégies d'apprentissage par renforcement pour traiter les causes profondes de ce problème.

Dans cette tâche sujette aux erreurs, le développement des systèmes auto adaptative pour aider le spécialiste de choisir la meilleur action. C'est pourquoi l'utilisation d'autres techniques comme les méthodes d'apprentissage automatique lequel vise à fournir des techniques et des méthodes pour accumuler, modifier et mettre à jour les connaissances dans les systèmes informatiques, et en particulier des mécanismes pour aider le système à induire des connaissances à partir d'exemples ou de nouvelles données. Ces méthodes sont appropriées lorsque nous n'avons pas de solutions algorithmiques, en l'absence de modèles formels, ou lorsque nous ne connaissons Pas précisément le domaine d'application.

L'une des approches de l'apprentissage automatique est l'apprentissage par renforcement, qui met l'accent sur l'apprentissage de l'individu par le biais d'interactions avec son environnement, contrairement aux approches classiques de l'apprentissage automatique qui privilégient l'apprentissage auprès d'un enseignant bien informé, ou sur le raisonnement à partir d'un modèle complet de l'environnement.

Dans l'apprentissage par renforcement, l'apprenant ne se voit pas dire quelle action il doit entreprendre, mais doit plutôt trouver quelles actions lui rapportent une meilleure récompense après les avoir essayées. Les caractéristiques les plus distinctives de l'apprentissage par renforcement sont la recherche par essais et erreurs et la récompense différée.

L'objectif de ce mémoire est d'améliorer la chaine logistique des documents avec la technique d'apprentissage du renforcement pour obtenir les données nécessaires à son exécution à partir de l'activité "Etudier le dossier", renvoyant ainsi la combinaison optimale (état, action) pour le traitement du dossier de pension en cours.

Chapitre1

Présentation du Projet

Ce chapitre traite une description générale du sujet et notre problématique puis présentera notre objectif et notre démarche de résolution par l'apprentissage par renforcement.

I. Contexte général :

Dans le cadre de l'étude du régime de retraite marocaine ; plus précisément notre cas d'étude: la caisse de retraite marocaine MPF dans le régime civile selon le budget général, on peut conclure que le grand défi c'est la nécessité de faire durer ce système plus longtemps et la difficulté d'offrir un service (la pension de retraite) au bon moment (premier mois de sa retraite) ainsi de rendre auto adaptative en temps réels . Pour cela nous avons implémenté l'algorithme Q_learning un modèle d'apprentissage par renforcement lequel connecter à l'activité « exécution d'un système de recommandation » afin de renvoyer la combinaison optimal de chaque (état, action) et aussi de déterminer le meilleur chemin du processus des documents de retraite.

II. Objectif d'intégrer l'apprentissage automatique aux processus métiers

Pour les entreprises, l'Intelligence Artificielle et le Machine Learning représentent une réelle piste d'amélioration des processus métiers avec en plus, une meilleure satisfaction des collaborateurs, une réduction du taux d'erreurs et des coûts.

L'idée est d'automatiser certaines tâches quotidiennes qui impliquent beaucoup de temps et d'argent pour un gain relativement faible. Le temps étant une ressource de plus en plus rare et convoitée, on cherche alors à l'optimiser par tous les moyens. Une des manières de le préserver est de déléguer certaines tâches rébarbatives à la technologie. La technologie intervient ici pour injecter de l'intelligence aux processus métiers et éliminer les processus manuels. Le but étant de passer le moins de temps possible sur les tâches chronophages et à faible valeur ajoutée pour les automatiser.

Le Machine Learning offre ici des opportunités formidables de simplifier la vie aux utilisateurs pour concilier le besoin de simplicité et d'efficacité avec en plus une meilleure gestion des services, conformité et réduction des coûts.

III. Séparation du problème

Afin de bien concevoir le problème à résoudre, il est d'abord primordial de pouvoir séparer les différents modules d'implantation au sein de notre service web. En effet l'exécution d'un système de recommandation se sépare en trois sous-modules principaux :

- Le module noyau c'est la partie théorique où nous avons analysée le sujet, déterminer les méthodes mathématiques lequel les variances du Q_learning pour bien choisir le modèle d'apprentissage par renforcement convenable de notre cas d'étude.
- Le module fonctionnel c'est la partie d''exploration de notre environnement ; la chaine logistique de retraite par les outils de modélisation SCOR et BPMN en utilisant la modèle DMAIC de Six Sigma pour faire l'évaluation de notre Processus.
- Le module de développement c'est la partie de résolution technique où nous avons se fixer à choisir les outils nécessaires de réalisation et voir les résultats pour l'interpréter.

Chapitre2

L’évolution de Q_Learning et le processus de décision de Markov

Ce chapitre a pour objectif de monter tout d’abord le principe de l’apprentissage par renforcement, pour montrer ensuite son évolution et son relation avec le processus de décision de Markov.

I. L'apprentissage par renforcement

L'apprentissage par renforcement est un type d'apprentissage automatique qui est influencé par la psychologie comportementaliste. Il s'agit de savoir comment les agents logiciels doivent agir dans un environnement afin de maximiser une certaine notion de récompense cumulative. Elle n'utilise aucun ensemble de données de formation pour apprendre le schéma. L'apprenant n'est pas informé des actions à entreprendre, comme dans la plupart des formes d'apprentissage automatique, mais doit plutôt découvrir quelles sont les actions les plus gratifiantes en les essayant. Dans les cas les plus intéressants et les plus difficiles, les actions peuvent affecter non seulement la récompense immédiate, mais aussi la situation suivante et, par-là, toutes les récompenses ultérieures. Ces deux caractéristiques : la recherche par essais et erreurs et la récompense différée sont les traits distinctifs du l'apprentissage par renforcement

Le modèle d'apprentissage par renforcement consiste en :

- Un ensemble d'environnement et d'agents indique S.
- Un ensemble d'actions A de l'agent.
- Des politiques de transition des états aux actions.
- Des règles qui déterminent la récompense scalaire immédiate d'une transition.
- Des règles qui décrivent ce que l'agent observe.

Une tâche est définie par un ensemble d'états, s∈S, un ensemble d'actions, a∈A, une fonction de transition état-action,

T : S×A→S, et une fonction de récompense, R : S×A→R. À chaque étape, l'apprenant (également appelé l'agent) sélectionne une action, puis, en conséquence, reçoit une récompense et son nouvel état. L'objectif de l'apprentissage par renforcement est d'apprendre une politique, une cartographie des états aux actions, Π : S →A qui maximise la somme de sa récompense au fil du temps.

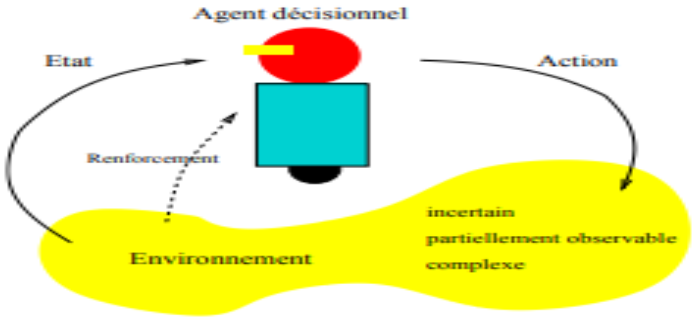

Figure 1:Schéma de fonctionnement de l'apprentissage par renforcement

Dans l'apprentissage automatique, l'environnement est formulé comme un processus de décision de Markov (MDP), car de nombreux algorithmes d'apprentissage de renforcement pour ce contexte utilisent des techniques de programmation dynamique.

II. L'évolution de Q_Learning et son relation avec le processus de décision de Markov.

1. Le processus de décision de Markov

Le processus de décision de Markov (MDP) est une méthode mathématique utilisé pour modéliser la prise de décision dans les situations où la cible est en partie aléatoire et en partie sous le contrôle d'un décideur.

Les MDP sont utiles lorsque nous étudions un large éventail de processus d'optimisation qui peuvent être résolus par la programmation dynamique et l'apprentissage par renforcement. Les MDP consistent en un ensemble fini d'états, des fonctions de valeur pour ces états, un ensemble fini d'actions, une politique et une fonction de récompense.

L'idée générale de cette situation est que nous sommes un agent dans un certain monde, et que nous avons un ensemble d'états, d'actions (pour chaque état), une probabilité de transition (que nous ne connaissons peut-être pas réellement) et une fonction de récompense. L'objectif d'un agent rationnel est de maximiser la somme attendue des récompenses (actualisées) pour un état où $0 \leq \gamma \leq 1$ $\mathbb{E}\left[\sum_t \gamma^t R\left(s_t\right)\right]$ est notre facteur d'actualisation, que nous supposons généralement égal à un pour les cas de jouets uniquement.

Il s'agit de la mise en place d'un processus de décision de Markov (PDM), avec la contrainte supplémentaire que la probabilité de passer à un autre état ne dépend que de l'état actuel, c'est pourquoi nous appelons cela un processus de décision de Markov. Puisque le but ultime est de maximiser l'utilité attendue, nous devrons apprendre une fonction politique π qui fait correspondre les états aux actions. Enfin, bien que cela ne soit pas strictement nécessaire, il est courant de "pimenter" le problème du PDM en supposant que les actions sont non déterministes.

Afin de comprendre comment obtenir une politique optimale, nous devons d'abord nous rendre compte de la manière de définir exactement la valeur V*(s) d'un état, qui, en d'autres termes, est l'utilité attendue que nous obtiendrons si nous sommes à l'état s et jouons de manière optimale. Il existe, deux formulations courantes pour V*(s). La première, de R & N suppose que l'aspect récompense de la formule est défini en termes d'état uniquement :

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s,a,s') V^*(s')$$

La deuxième formulation, que la classe CS 188 de Berkeley utilise1, définit la récompense en termes de triple état-action-successeur :

$$V^*(s) = \max_a \sum P(s,a,s') [R(s,a,s')+\gamma V^*(s')]$$

Ces deux ensembles d'équations peuvent être appelés les équations de Bellman, qui caractérisent les valeurs optimales mais nous aurons généralement besoin d'une autre façon de les calculer, comme nous le montrerons dans la suit.

Il est également courant de définir une nouvelle quantité appelée valeur Q en ce qui concerne les paires état-action :

$$Q^*(s,a) = \sum_{s'} P(s,a,s') [R(s,a,s')+\gamma V^*(s')]$$

En d'autres termes, Q(s,a) est l'utilité attendue à partir de l'état s, en prenant l'action a, et par la suite, en jouant de manière optimale. Nous pouvons donc mettre en relation les V et les Q avec l'équation suivante :

$$V^*(s)=\max_a Q^*(s,a)$$

Les nœuds "normaux" correspondent à Vs, et les nœuds "aléatoires" correspondent à Qs. Les deux nœuds ont de multiples successeurs : les V parce que nous avons le choix des actions, et les Q parce que, même si nous nous engageons dans une action, le résultat réel est généralement non déterministe, de sorte que nous ne saurons pas dans quel état nous nous retrouverons.

Pour la politique optimale ; Il existe deux méthodes

L'itération de la valeur est un algorithme itératif qui calcule les valeurs des états indexés par un k, comme dans $V_k(s)$, qui peut également être pensé comme la meilleure valeur d'un état s en supposant que le jeu se termine par k pas de temps. Il ne s'agit pas de la politique proprement dite, mais ces valeurs sont utilisées pour déterminer la politique optimale2.

En commençant par $V_0(s)=0$ pour tous les s, l'itération des valeurs effectue la mise à jour suivante et elle se répète jusqu'à ce qu'on lui dise d'arrêter :

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s,a,s') [R(s,a,s')+\gamma V_k(s')]$$

Pour rendre cela intuitif, à chaque étape, nous avons un vecteur de valeurs $V_k(s)$, puis nous faisons un calcul pour obtenir le vecteur suivant. Notez qu'on pourrait calculer toutes les valeurs dont nous avons besoin, mais cela prendrait trop de temps en raison des arbres d'état de profondeur répétés et infinis.

Pour prouver que l'itération de la valeur converge vers V* (et de manière unique), on fait appel à la contraction et au fait que γ^k, tant que $\gamma \neq 1$, descendra à zéro

Politiques de transition est généralement une amélioration par rapport à l'itération des valeurs, car les politiques convergent souvent bien avant les valeurs, de sorte que nous alternons entre les étapes d'évaluation et d'amélioration des politiques. Dans la première étape, nous supposons qu'on nous donne une politique π et que nous devons déterminer les utilités attendues de chaque état lors de l'exécution de π. Ces valeurs sont caractérisées par les équations suivantes :

$$V^{\pi}(s) = \sum_{S'} P(s,\pi(s),s') \, [R(s,\pi(s),s') + \gamma \, V^{\pi}(s')]$$

Notez que l'opérateur MAX est parti car π(s) nous donne notre action. Ainsi, nous pouvons résoudre <ces équations en temps $O(n^3)$ avec des méthodes de multiplication matricielle standard3.

Il est en fait assez facile d'améliorer les politiques à partir des valeurs Q :

On peut aussi utiliser le formulaire expectimax, plus compliqué :

$$\pi*(s) = arg_a \max \sum_{S'} P(s,a,s') \, [R(s,a,s') + \gamma V^*(s')]$$

Les deux étapes se succéderont jusqu'à la convergence, et comme l'itération des valeurs, l'itération des politiques est optimale. En outre, ces équations sont en réalité de l'extraction de politiques, dans le sens où, compte tenu des valeurs, nous savons comment obtenir une politique. Comparaison des deux algorithmes :

En général, l'itération de la politique semble être un peu mieux que l'itération des valeurs, bien que je suppose que cette dernière pourrait être plus simple à mettre en œuvre dans certains cas puisqu'elle n'implique que la partie de mise à jour des valeurs. Si nous ne savons pas exactement dans quel état nous sommes, nous n'avons pas de notion de s ici, nous devons donc changer notre représentation du problème. C'est le cas du processus de décision Markov partiellement observable (POMDP).

Nous complétons le POMDP avec un modèle de capteur P(e| s) et traitons les états comme des états de croyance. Dans un MDP discret à n états, le vecteur d'état de croyance b serait un vecteur à n dimensions avec des composantes représentant les probabilités d'être dans un état particulier.

2. L'évolution de Q_Learning

Vu la complexité des algorithmes présentés, de nombreux travaux se sont penchés sur l'utilisation de méthodes permettant d'accélérer ces calculs. L'utilisation d'apprentissage par renforcement. Ce type d'algorithmes peut être utilisé non seulement pour apprendre la politique optimale, mais aussi pour apprendre la fonction de transition si celle-ci n'est pas connue. Parmi les algorithmes d'apprentissage par renforcement, nous pouvons citer le Q-learning [Watkins et Dayan, 1992] qui est le plus connu. . Avant de rentrer dans les détails, il est plus facile de nous imaginer dans le même cadre de PDM, sauf que nous devons apprendre notre politique en ligne, et non hors ligne. C'est plus difficile car nous supposons généralement que nous ne connaissons pas à l'avance les probabilités de transition, ni même les récompenses.

Tout d'abord, considérons le cas du renforcement passif, où l'on nous donne une politique fixe éventuellement des déchets, π et où le seul but est d'apprendre les valeurs à chaque état, selon les équations de Bellman.

Voici une façon d'obtenir les valeurs :

Apprendre les transitions P et les récompenses R par des essais, puis appliquer immédiatement l'étape d'évaluation de la politique dans l'itération de la politique. Pour rappel, cela permet de calculer les valeurs (comme dans l'itération des valeurs) mais comme la politique est fixe, nous pouvons résoudre rapidement l'ensemble des équations.

Cette technique, que Russell et Norvig semblent appeler la programmation dynamique adaptative, fait partie de la classe des techniques d'apprentissage basées sur des modèles, car cela suppose que nous devons calculer un modèle de l'environnement (c'est-à-dire les transitions et les récompenses). Ici, notre agent pourrait jouer une série d'actions à partir de différents états ("redémarrage" si nécessaire). À la fin, il disposera de données sur les récompenses obtenues, ainsi que sur le nombre de fois qu'il est passé de l'état i à l'état j.

Pour calculer la fonction de transition, nous convertirons les chiffres en moyennes de manière à obtenir une "estimation de probabilité maximale". Les résultats sont plus faciles à obtenir car nous pouvons voir $R(s,a,s')$ immédiatement lorsque nous passons de s à s'. Dans l'apprentissage sans modèle, nous ne prenons pas la peine de calculer le modèle de transition, mais nous calculons seulement les valeurs réelles de chaque état. Il existe une technique de base appelée estimation directe de l'utilité qui entre dans cette catégorie de méthodes. L'idée ici est que pour chaque essai, nous devons enregistrer la somme des récompenses actualisées pour chaque état visité. Après une série d'essais, ces valeurs vont s'équilibrer et converger, bien que lentement car l'estimation directe de l'utilité ne tient pas compte des

équations de Bellman,il nous faut une meilleure solution, puisque nous avons déjà une politique, nous aimerions faire une certaine forme d'évaluation de la politique, mais nous n'avons pas les transitions et les récompenses. Nous pouvons les approximer avec nos échantillons à la place, ce qui conduit à un apprentissage par différence temporelle. Il s'agit d'une tactique sans modèle où, après chaque transition (s,a,s′,r), nous mettons à jour les valeurs :

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s)+\alpha[R(s,\pi(s),s')+\gamma V^{\pi}(s)]$$

L'apprentissage fonctionne en ajustant les estimations de l'utilité vers l'équilibre idéal tel qu'énoncé par les équations de Bellman. Mais que faire si nous voulons obtenir une nouvelle politique ? L'apprentissage TD n'apprend pas les probabilités de transition, donc nous passons à l'apprentissage des valeurs Q, car il est plus facile d'extraire des actions à partir des valeurs Q. Cela nous amène maintenant à l'apprentissage par renforcement actif, où nous devons apprendre une politique optimale en choisissant des actions. Il est clair qu'il y aura des compromis entre l'exploration et l'exploitation. La solution ici est un algorithme appelé Q-Learning, qui calcule itérativement les valeurs Q :

$$Q(s,a)\leftarrow(1-\alpha)Q(s,a)+\alpha[R(s,a,s')+\gamma \max Q(s',a')]$$

Chaque échantillon est calculé à partir d'un tuple (s,a,s′,r). Le Q-Learning convergera vers une solution optimale même si les actions que nous entreprenons sont constamment sous-optimales

Il serait probablement judicieux de disposer de bonnes heuristiques pour décider des actions à entreprendre, ce qui pourrait faire converger plus rapidement l'apprentissage par la qualité. La stratégie la plus simple (et la pire) : agir selon une politique en vigueur, mais avec une faible probabilité, nous choisissons une action aléatoire. Pour améliorer cela, utilisez des fonctions d'exploration qui pèsent l'importance des actions et des états selon que leurs valeurs sont établies ou non.

Cette information se propage en retour, de sorte que nous finissons par favoriser les actions, cela signifie que pour certaines fonctions d'exploration f, l'action que nous choisissons à une étape sera $arg_{a'}$ max f(Q(s′,a′),N(s′,a′)) où N représente les états, et la nouvelle mise à jour de Q-Learning est :

$$Q(s,a)\leftarrow(1-\alpha)Q(s,a)+\alpha[R(s,a,s')+\gamma \max_{a'} f(Q(s',a'),N(s',a')]$$

Il existe un proche parent du Q-Learning appelé SARSA, State-Action-Reward-State-Action. La mise à jour est, suite à la notation de Russell et Norvig de faire fixer les récompenses à un état :

$$Q(s,a)\leftarrow(1-\alpha)Q(s,a)+\alpha(R(s)+\gamma Q(s',a'))$$

Cela signifie que nous avons déjà une action a′ choisie pour l'État successeur s′. Q-Learning choisira la meilleure action a′, mais dans le SARSA, c'est fixe, et nous pouvons alors mettre à jour les Q(s,a)

3. Limite de Q-Learning:

SARSA et Q-Learning (la première version ici, pas la deuxième avec f) sont les mêmes pour un agent avide qui choisit toujours la meilleure action. Lorsque l'exploration a lieu, Q-Learning tente de choisir la meilleure action, mais SARSA choisit ce qui va réellement se passer. Cela signifie que le Q-Learning ne prête pas attention à la politique, c'est-à-dire qu'il est hors politique, mais le SARSA le fait, donc il est sur la politique.

Il convient de souligner que toute cette discussion précédente sur l'apprentissage de toutes les valeurs Q(s,a) n'est réellement applicable que pour le plus petit des problèmes, car nous ne pouvons pas comptabiliser tous ceux qui se posent dans des situations réelles. Nous avons donc recours à l'apprentissage approximatif des Q. Ici, nous utilisons des représentations basées sur les caractéristiques en représentant un état comme un vecteur de caractéristiques :

$$\mathbf{V(s) = w1\ f1(s) + \cdots + wn\ fn(s)}$$

Et il en va de même pour les valeurs Q :

$$\mathbf{Q(s,a) = w1\ f1(s,a) + \cdots + wn\ fn(s,a)}$$

Nous avons l'habitude de rendre les choses linéaires en termes de caractéristiques pour plus de simplicité. Cela présente l'avantage supplémentaire de pouvoir généraliser aux États que nous n'avons pas encore visités (en plus des avantages évidents de la compression). Notez que lorsque nous faisons la mise à jour de Q-Learning ici, nous changeons les pondérations. Chaque mise à jour des poids ressemble au même type de mise à jour que celle que nous voyons dans la descente stochastique du gradient.

Q-Learning est souvent très complexe et l'obtention de la politique optimale peut être très lente , la preuve de convergence de l'algorithme nécessite que chaque paire (état, action) ait été testée une infinité de fois.

Chapitre3

Les variances du Q_Learning dans l'apprentissage par renforcement

Ce chapitre Ce chapitre présente les différents algorithmes de l'apprentissage par renforcement pour justifier le choix du Q_learning en montrant leurs principes et leurs pseudocodes.

I. Programmation dynamique

La résolution des MDP a toujours été étroitement liée à l'idée de programmation dynamique,. Une approche intéressante qui associe la programmation dynamique à d'autres méthodes de contrôle discret systèmes d'événements. Il a également été largement utilisé dans les applications de contrôle optimal. Deux méthodes classiques de programmation dynamique pour les MDP sont l'itération de valeur et l'itération de politique.

1. Itération de la valeur

Comme indiqué précédemment, un moyen de trouver une politique optimale consiste à calculer fonction de valeur optimale. L'itération de valeur est un algorithme pour déterminer
Fonction, dont on peut prouver qu'elle converge vers les valeurs optimales de V. Voici le pseudo code de ce modèle:

Initialize $V(s)$ arbitrarily

repeat
  $\delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$
    $\delta \leftarrow \max(\delta, |v - V(s)|)$
  end for
until $\delta < \epsilon$

2. Itération de la politique

Une fois qu'une politique π a été évaluée en donnant la fonction de valeur Vπ , puis a été améliorée pour donner une politique π', on peut à nouveau appliquer l'évaluation sur la politique π' et l'améliorer, et ainsi de suite. On peut donc obtenir une séquence de politiques croissantes et de fonctions valeur. Il est garanti que chaque politique est une stricte amélioration de la précédente. Parce qu'un PDM fini a un nombre fini de politiques possibles, cette suite de politiques converge vers la politique optimale V * en un nombre fini d'itérations. L'algorithme « itération de politique » correspondant à ce processus. L'algorithme d'itération converge en un nombre d'itérations étonnamment faible.

Initialize $V(s)$ and $\pi(s)$ arbitrarly

repeat

Policy Evaluation
Solve the system of linear equations
$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s')\right]$

Policy Improvement
$optimalPolicy? \leftarrow TRUE$
for all $s \in S$ **do**
$b \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$
if $b \neq \pi(s)$ **then**
$optimalPolicy? \leftarrow FALSE$
end if
end for

until $optimalPolicy? = TRUE$

Un désavantage de l'algorithme d'itération de politique est le nombre d'itérations nécessaires pour faire converger l'évaluation d'une politique vers son évaluation optimale.

II. Algorithme Rmax:

Dans cette partie, nous présentons Rmax, un algorithme d'apprentissage par renforcement très simple utilisant un modèle de l'environnement [Brafman & Tennenholtz 2002]. Dans Rmax, l'agent maintient un modèle de l'environnement (c'est-à-dire les probabilités de transitions P a ss' et les retours Ra ss' du PDM). Ce modèle est inexact au début mais se raffine au fur et à mesure. L'agent agit toujours optimalement par rapport à ce modèle en utilisant l'algorithme "Value Iteration" (VI) pour calculer V et Q. Théoriquement VI est appelé à chaque fois que ce modèle est modifié. Ainsi, V et Q sont les fonctions de valeurs optimales par rapport au modèle de l'environnement. Le modèle de l'environnement est initialisé de manière optimiste: tous les retours Ra ss' du PDM sont initialisés au retour maximal. Pendant l'exécution de Rmax, les retours observés viennent remplacer les retours du modèle.

Rmax est une amélioration de Value Iteration car il est « online ». Value Iteration est un algorithme « off-line » au sens ou le calcul de V et Q est fait une fois pour toute avant utilisation de V et Q par l'agent. Dans Rmax, l'agent commence à agir dans l'environnement et la calcule de V et Q se fait peu à peu au fur et à mesure que l'agent agit dans l'environnement. Rmax résout le dilemme exploitation - exploration. Un problème à résoudre pour un agent apprenant par renforcement est le dilemme entre

des actions inconnues et exploiter des actions connues. A un instant donné, lorsque l'espace des états n'est pas complètement exploré, les retours R a ss' des transitions allant vers des états non explorés valent Rmax. Ainsi VI calcule une politique optimale pour aller vers ces états inconnus à grands retours, donc Rmax explore. De plus, Rmax explore optimalement. On dit que le dilemme exploitation – exploration est résolu implicitement par Rmax. En effet, Rmax marche toujours de la même manière: optimalement par rapport au modèle de l'environnement et n'a pas de mécanisme explicite pour dire je suis en train d'explorer ou d'exploiter. Rmax suit le biais de l'optimisme dans l'incertain. Par défaut, Rmax va vers les états qu'il ne connaît pas en supposant qu'ils sont bons. Rmax est simple. Cf code au paragraphe suivant.

Rmax est général: il s'applique non seulement à des PDM mais aussi à des jeux stochastiques. (Un jeu stochastique est un PDM multi-agent: plusieurs agents interagissent dans le graphe d'états. L'état suivant est déterminé par l'action jointe: combinaison des actions des agents.)

Algorithme Initialisation:

Entrée: longueur de la politique = T

Construction du modèle M' constitué des N états réels G1, G2, ..., GN, plus un état fictif G0. Chaque état possède:

Une valeur d'état V.

Une matrice d'actions jointes contenant dans chaque cellule:

Le retour observé initialisé avec Rmax.

La liste des états suivants rencontrés avec pour chaque état suivant

le nombre de fois où la transition a été effectuée.

(liste initialisée avec G0, avec une unique transition fictive effectuée).

Un attribut booléen « connu » ou pas, initialisé a faux.

Une valeur d'action jointe Q.

Exécution :

Répéter (C) Calcule la politique optimale de longueur T depuis l'état courant,

Exécute cette politique pendant T pas de temps.

Après chaque action jointe et transition vers un état suivant,

mettre à jour le retour observé

incrémenter le compteur de cette transition.

Si le compteur dépasse un seuil, alors

l'action jointe est connue aller en (C)

III. Algorithme du Monte Carlo

Dans cette partie, nous voyons comment associer l'idée de la programmation dynamique avec l'idée de Monte-Carlo (MC). Le terme Monte-Carlo consiste à effectuer des simulations au hasard, à retenir les résultats des simulations et à calculer des moyennes de résultats. La stratégie du Monte Carlo est adaptée lorsque l'on ne connaît pas le modèle du domaine (les probabilités de transitions P a ss' et les retours Ra ss' du PDM). C'est une stratégie simple. Il faut que les tâches de l'agent soient décomposées en épisodes, et un épisode correspondra à une simulation. Dans ce chapitre, à partir d'une politique π, nous regardons comment obtenir une estimation de la fonction de valeur d'états Vπ , puis comment estimer une fonction de valeur d'actions Qπ , puis comment améliorer une politique (ce qu'on appelle le « contrôle Monte Carlo »).

Dans cette partie, on suppose, la politique π donnée et on cherche à l'évaluer avec la fonction de valeur d'état Vπ. La stratégie consiste à lancer des épisodes dans lequel l'agent utilise la politique π. Dans un épisode, pour chaque état, on enregistre le résultat de l'épisode et on dit que V(s) est la moyenne des résultats enregistrés. V(s) converge vers Vπ (s). La loi des grands nombres dit que l'écart-type autour de la moyenne décroît en 1/n1/2 où n est le nombre de résultats moyennés. Le pseudocode de l'évaluation MC d'une politique est :

Initialize:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
(a) Generate an episode using π
(b) For each state s appearing in the episode:
$R \leftarrow$ return following the first occurrence of s
Append R to $Returns(s)$
$V(s) \leftarrow \text{average}(Returns(s))$

IV. Apprentissage par différence temporelle

1. TD (0): estimation de la fonction valeur

Les méthodes de différence temporelle partagent une caractéristique commune:
valeur itération ou évaluation des politiques, ils utilisent les estimations pour l'État,valeurs d'autres états pour mettre à jour l'estimation de l'état actuel, qui est généralement connu sous le nom d'amorçage. La méthode de différence temporelle la plus simple TD (0) a été introduit par (Sutton, 1988) et à la règle de mise à jour suivante:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

L'état s' est observée et la récompense r collectée après l'application de la politique π
à l'état s. Cette méthode fonctionne comme l'étape d'évaluation des politiques dans le sens
qu'il estime la fonction de valeur pour une politique donnée mais n'améliore pas la politique.
L'idée derrière cette procédure est que la quantité r + V (s') agit comme
estimation pour V (s) et, probablement, il est plus proche de la valeur de l'État, pour la politique π, car il est calculé sur la base d'une récompense immédiate, intégrant connaissances de l'environnement dans l'estimation. En fait, le terme pesait par α n'est rien de plus qu'une estimation de l'erreur entre l'état actuel fonction de valeur et la fonction de valeur d'état réel, pour la politique donnée. Cette procédure peut être considérée comme une sorte d'approximation stochastique, bien que la plupart des preuves de convergence pour les méthodes d'apprentissage par différence temporelle ont été faites sans le relier à la théorie derrière l'approximation stochastique.
La première initiative visant à rapprocher les deux ensembles de connaissances a été
(Tsitsiklis, 1994).
Ce type de procédure est également censé faire un exemple de sauvegarde car la nouvelle estimation de la fonction de valeur est obtenue sur la base d'un échantillon des options possibles pour l'état suivant. Les algorithmes de programmation dynamique, d'autre part, sont censés effectuer une sauvegarde complète, en s'appuyant sur les estimations de tous les États successeurs possibles.
Pour que la méthode converge, elle doit visiter chaque état infiniment souvent et
le taux d'apprentissage α doit être lentement diminué, ce qui a été prouvé dans (Sutton, 1988).

Algorithm 2.3 TD(0) learning

Initialize $V(s)$ arbitrarly
Choose the policy π to be evaluated

Initialize s
loop
$a \leftarrow$ probabilistic outcome of pdf $\pi(s)$
Take action a, observe reward r and next state s'
$V(s) \leftarrow V(s) + \alpha\,(r + \gamma V(s') - V(s))$
$s \leftarrow s'$
end loop

2. Q_learning

L'une des avancées les plus importantes en AR est l'algorithme Q-learning de Watkins. La règle de mise à jour de Q-Learning est:

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a\,[\, r_{t+1} + g \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
Initialize s
Repeat (for each step of episode):
Choose a from s using policy derived from Q (e.g., ε-greedy)
Take action a, observe r, s'
$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
$s \leftarrow s'$;
until s is terminal

On remarque que la politique est absente de cet algorithme car elle est implicitement dictée par Q. En ce sens QL est aussi « on-line ». Cependant, l'action choisie a' n'est pas forcément l'action optimale correspondant à la valeur Q utilisée pour la mise à jour.

On remarque aussi que le choix de l'action est encore ε-greedy, sinon la méthode ne marche pas. Si l'on ne souhaite pas avoir de choix ε-greedy, on peut initialiser les valeurs Q avec de *très grandes* valeurs.

3. Sarsa

Sarsa a été introduit pour la première fois par (Rummery et Niranjan, 1994) comme une méthode on policy pour apprendre la politique optimale tout en contrôlant le MDP.

Dans cette situation, l'algorithme se comporte selon la même politique qui est en cours d'amélioration et, par conséquent, la règle de mise à jour est la suivante:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\, r + \gamma Q(s\,0\,, a0\,) - Q(s, a)$$

La méthode repose sur des informations sur les variables s, a, r, s', a' et cela explique l'origine du nom Sarsa, introduit par (Sutton, 1996). Les résultats de la convergence sont visibles dans (Singh et al, 2000) et, fondamentalement, exiger que chaque paire état-action soit visitée infiniment souvent et que la politique utilisée converge vers une politique gourmande. Exemples de politiques peut être utilisé pour répondre à l'exigence peut être vu dans la section .La forme de l'algorithme est présentée dans l'algorithme. Notez que la politique utilisé pour choisir les actions est le même que celui utilisé pour l'évaluation et amélioration. Comme indiqué précédemment, dans l'algorithme Q-learning, la politique utilisée pour l'évaluation et l'amélioration était gourmand, ce qui est identifié par l'utilisation de l'opérateur max dans le calcul de la nouvelle estimation pour Q (s, a).

Algorithm 2.5 Sarsa

Initialize $Q(s, a)$ arbitrarly

Initialize s
$a \leftarrow$ probabilistic outcome of policy derived from Q
loop
  Take action a, observe reward r and next state s'
  $a' \leftarrow$ probabilistic outcome of policy derived from Q
  $Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma Q(s', a') - Q(s, a)\big)$
  $s \leftarrow s'$ and $a \leftarrow a'$
end loop

V. Exploration basée sur la différence de valeur

1. Politique E-greedy

Ce type de politiques s'inscrit dans la description générale de l'équation et est essentiellement un mélange d'une politique gourmande (pour l'exploitation) et d'une politique uniforme (pour l'exploration).Le paramètre peut être arbitrairement petit et conduire la convergence vers une politique gourmande (et optimale). L'équation décrit une telle politique.

$$\pi(s,a) = \begin{cases} \dfrac{\epsilon}{|A|} & \text{if } a \text{ is a greedy action,} \\ 1 - \epsilon + \dfrac{\epsilon}{|A|} & \text{otherwise.} \end{cases}$$

2. Politiques Softmax:

Bien que les politiques -grasses soient largement utilisées dans l'apprentissage par renforcement, elles ont l'inconvénient de donner un poids égal aux actions non gourmandes.

En fait, certains d'entre eux pourraient être incroyablement meilleurs que d'autres, bien qu'ils ne soient pas gourmands. Un moyen d'éviter ce problème consiste à utiliser une fonction utilitaire sur les actions pour mieux les distinguer. Ce type de méthodes, appelées softmax, donne toujours la plus forte probabilité à l'action gourmande mais ne traite pas toutes les autres de la même manière.

Dans le contexte d'apprentissage par renforcement, la fonction Q semble être une utilité naturelle à utiliser. La méthode softmax la plus courante utilisée dans l'apprentissage par renforcement repose sur une distribution de Boltzmann et contrôle l'accent mis sur l'exploration à travers un paramètre de température $\tau > 0$, tel que défini dans l'équation

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}$$

De la même façon que pour les politiques E-greedy, lorsque $\tau \to 0$, la politique devient gourmande rendre la méthode adéquate pour l'utilisation avec des algorithmes sur la politique.

VI. Traces d'éligibilité

1. TD (λ)

Il est possible d'étendre TD (0) avec des traces d'éligibilité en définissant les traces comme:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

où $0 \leq \lambda \leq 1$ est le paramètre contrôlant la décroissance. Avec ça mécanisme, chaque fois qu'un état est visité, son compteur est incrémenté de 1, après quoi il diminue de façon exponentielle.

L'idée de la méthode est que l'erreur de mise à jour de la règle dans l'équation est désormais affecté non seulement par α mais par l'éligibilité correspondante trace également, en utilisant l'expression suivante:

$$V(s) \leftarrow V(s) + \alpha\, \delta\, e(s)$$

ou

$$\delta = r + \gamma V(s\,0) - V(s)$$

Voici la version algorithmique de cette méthode

Initialize $V(s)$ arbitrarly and $e(s) = 0$, for all $s \in S$
Choose the policy π to be evaluated

Initialize s
loop
  $a \leftarrow$ probabilistic outcome of pdf $\pi(s)$
  Take action a, observe reward r and next state s'
  $\delta \leftarrow r + \gamma V(s') - V(s)$
  $e(s) \leftarrow e(s) + 1$
  for $\sigma \in S$ **do**
    $V(\sigma) \leftarrow V(\sigma) + \alpha\delta e(\sigma)$
    $e(\sigma) \leftarrow \gamma\lambda e(\sigma)$
  end for
  $s \leftarrow s'$
end loop

2. SARSA (λ)

L'idée de traces d'éligibilité peut également être appliquée à Sarsa en utilisant des traces pour chaque paire État-action et pas seulement pour chaque État.

La méthode a été explorée pour la première fois dans (Rummery et Niranjan, 1994 ; Rummery, 1995), bien que la convergence de la méthode pour un λ général soit encore à prouver.

Les équations des deux méthodes sont très similaires, sauf que Sarsa agit surValeurs Q. Les traces pour Sarsa(λ) peuvent être définies comme :

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{otherwise} \end{cases}$$

et la règle de mise à jour décrite par l'expression suivante :

$Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s)$ où $\delta = r + \gamma Q(s0, a0) - Q(s, a)$

et voici la version algorithmique de cette méthode:

Chapitre4

Modélisation du processus de retraite du MPF dans le régime civile selon le budget général

Dans ce chapitre on va présenter l'exploration de la chaine logistique de retraite par les outils de modélisation SCOR et BPMN en utilisant la modèle DMAIC de Six Sigma pour faire l'évaluation de notre Processus.

I. Exploration de la chaine logistique des documents de retraite par SCOR

La méthode SCOR est "une méthode de représentation des flux d'une entreprise permettant de modéliser ses différentes structures. En modélisant les systèmes logistiques, SCOR permet de créer un langage commun aux acteurs de la supply chain et d'harmoniser leurs pratiques.

Nous allons détailler notre problème selon trois niveaux :

1. Niveau stratégique

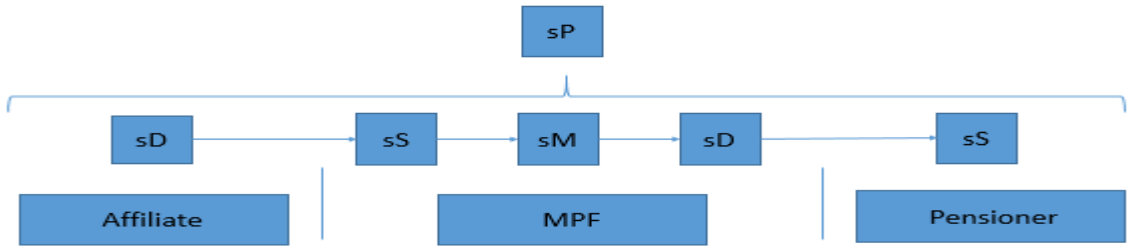

Figure 2: Le modèle de niveau stratégique du comité de retraite marocain

L'organisation étudiée est la MPF, son client est le retraité et son fournisseur est l'affilié cotisant. L'affilié gère le processus Deliver (sD) qui définit les règles de gestion de sa contribution au régime de retraite. La FPM gère trois processus, Source (sS) qui décrit la gestion de l'affiliation et de la cotisation ; Make (sM) qui décrit la gestion de la liquidation des droits, Deliver (sD) qui décrit la gestion du paiement du pensionné. Le retraité gère le processus Source (sS) qui décrit ses avantages d'une pension. Le processus Plan (sP) est celui qui équilibre l'offre et la demande globales pour développer un plan d'action, qui répond le mieux aux exigences d'approvisionnement, de production et de livraison.

2. Niveau tactique

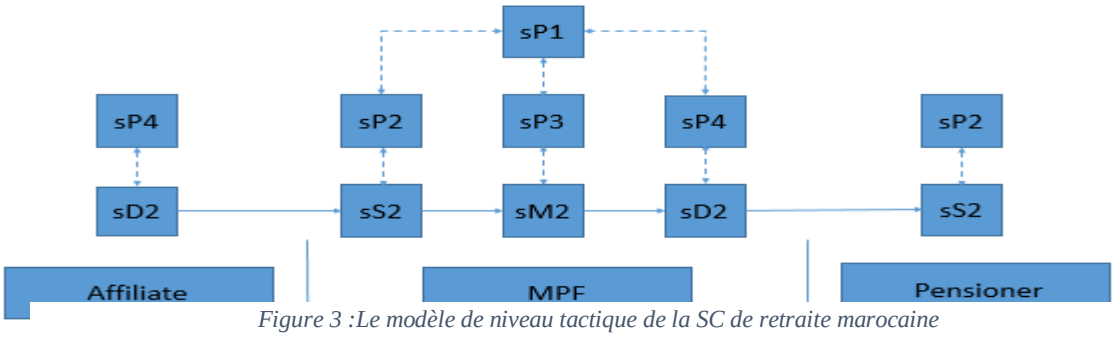

Figure 3 :Le modèle de niveau tactique de la SC de retraite marocaine

Le produit étudié (la pension de retraite) est un produit fabriqué sur commande, les processus de ce niveau sont donc : sS2 (Source Make-to-Order Product), sM2 (Make-to-Order Product), sD2 (Deliver Make-to-Order Product). Nous notons que les processus Plan indiqués dans la figure 2.2 sont sP1 (Plan Supply Chain), sP2 (Plan Source), sP3 (Plan Make), sP4 (Plan Deliver)

3. Niveau opérationnel :

Dans ce niveau on va spécifier les sous processus, donc on va détailler les processus SM2, SS2, SD2.

- Le sous processus SS2 :

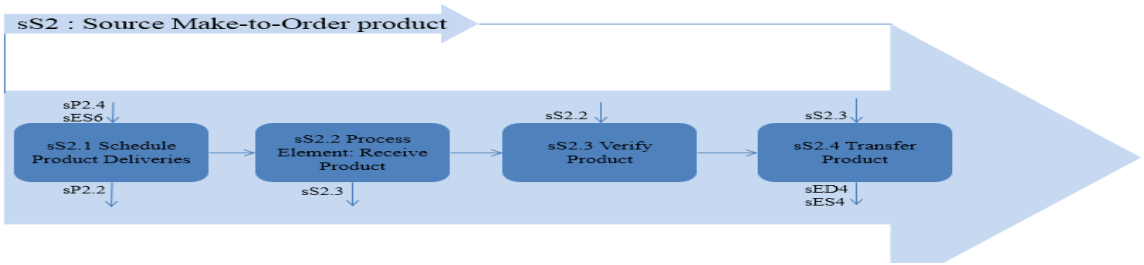

Figure 4:le sous processus make-to-ordre product

Dan Dans le cas présent, le processus d'affiliation et de contribution se compose de quatre éléments. sS2.1, sS2.2, sS2.3 et sS2.4. sS2.1 Livraisons de produits programmées correspond à la réception et à l'allocation des fichiers des contributeurs, il reçoit en entrée les informations des processus Plans de source sP2.4 et Sélection logistique sES6. En sortie, sS2.1 exécute le processus sP2.2 Identifier, évaluer et agréger les ressources de produits.

- Le sous processus SM2 :

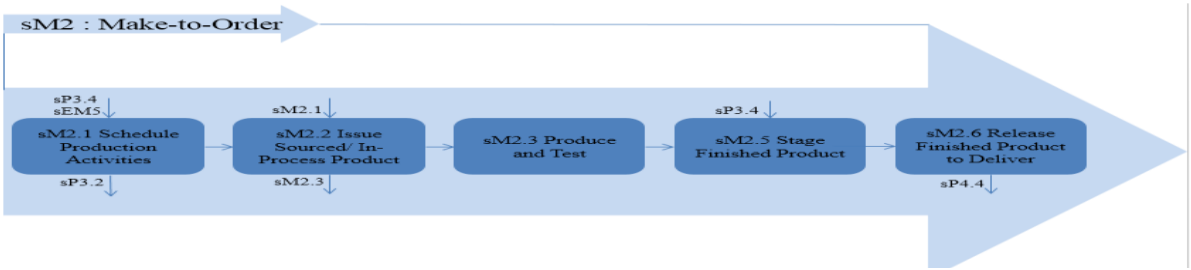

Figure 5 : le sous processus make-to-ordre

Dans cette étude de cas, le processus de liquidation des droits se compose de cinq éléments. sM2.1, sM2.2, sM2.3, sM2.5 et sM2.6. sM2.1 Planifier les activités de production correspond à la réception et à l'attribution des dossiers de liquidation, il reçoit comme information d'entrée des processus sP3.4 Établir les plans de production et sEM5 Gérer la fabrication des équipements et des installations. En sortie, sM2.1 exécute le processus sP3.2 Identifier, évaluer et agréger les ressources de production. sM2.2 Délivrer le produit source/en cours de fabrication correspond aux constitutions des droits. En entrée, sM2.2 reçoit les règles et les informations de calcul de sM2.1. En sortie, sM2.2 donne des

Informations en retour à sM2.3. sM2.3 Produire et tester correspond à la liquidation des droits. sM2.5 Étape Produit fini correspond à la concession des droits, à cette étape tous les fichiers liquidés sont édités dans une décision. En entrée, il reçoit les données et les fichiers issus de l'exécution de l'étape sP3.4 Établir les plans de production. sM2.6 Libérer le produit fini à livrer correspond à l'envoi des fichiers liquidés et concédés au service de paiement. En sortie, il donne le plan de paiement en exécutant le processus sP4.4 Établir les plans de livraison.

- Le sous processus SD2 :

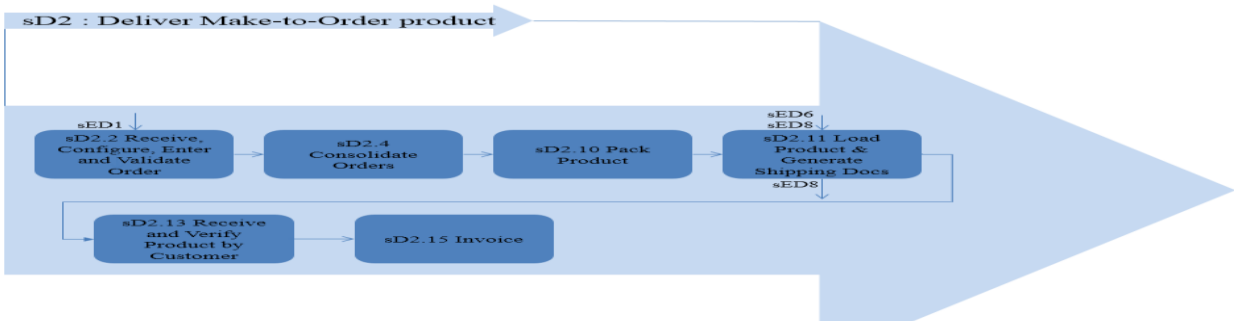

Figure 6 : le sous processus deliver make-to-ordre product

Dans notre étude de cas, le processus de paiement se compose de six éléments. Nous avons donc choisi parmi les sept éléments les six qui correspondent fonctionnellement à notre cas, et qui sont sD2.2, sD2.4, sD2.10, sD2.11, sD2.13 et sD2.15. sD2.2 Recevoir, configurer, entrer et valider l'ordre correspond aux processus de réception, de contrôle et d'intégration.

II. Evaluation de la chaine des documents de retraite par BPMN en utilisant six sigma :

- BPMN Business Process Modeling Language: Première initiative de modélisation des processus Métiers par le biais d'une syntaxe XML. par nécessité d'avoir un langage graphique... Ces notations créées par un regroupement d'une trentaine d'organisations œuvrant dans la modélisation des processus d'affaires.
- Six sigma : La méthode Six Sigma, orientée qualité, vise à réduire la variabilité d'un processus pour tendre vers le zéro défaut. La méthode Six Sigma se base sur une démarche fondée à la fois sur la **voix du client** (enquêtes, etc.) et sur des **données mesurables** (indicateurs, etc) et fiable

1. Définir la phase

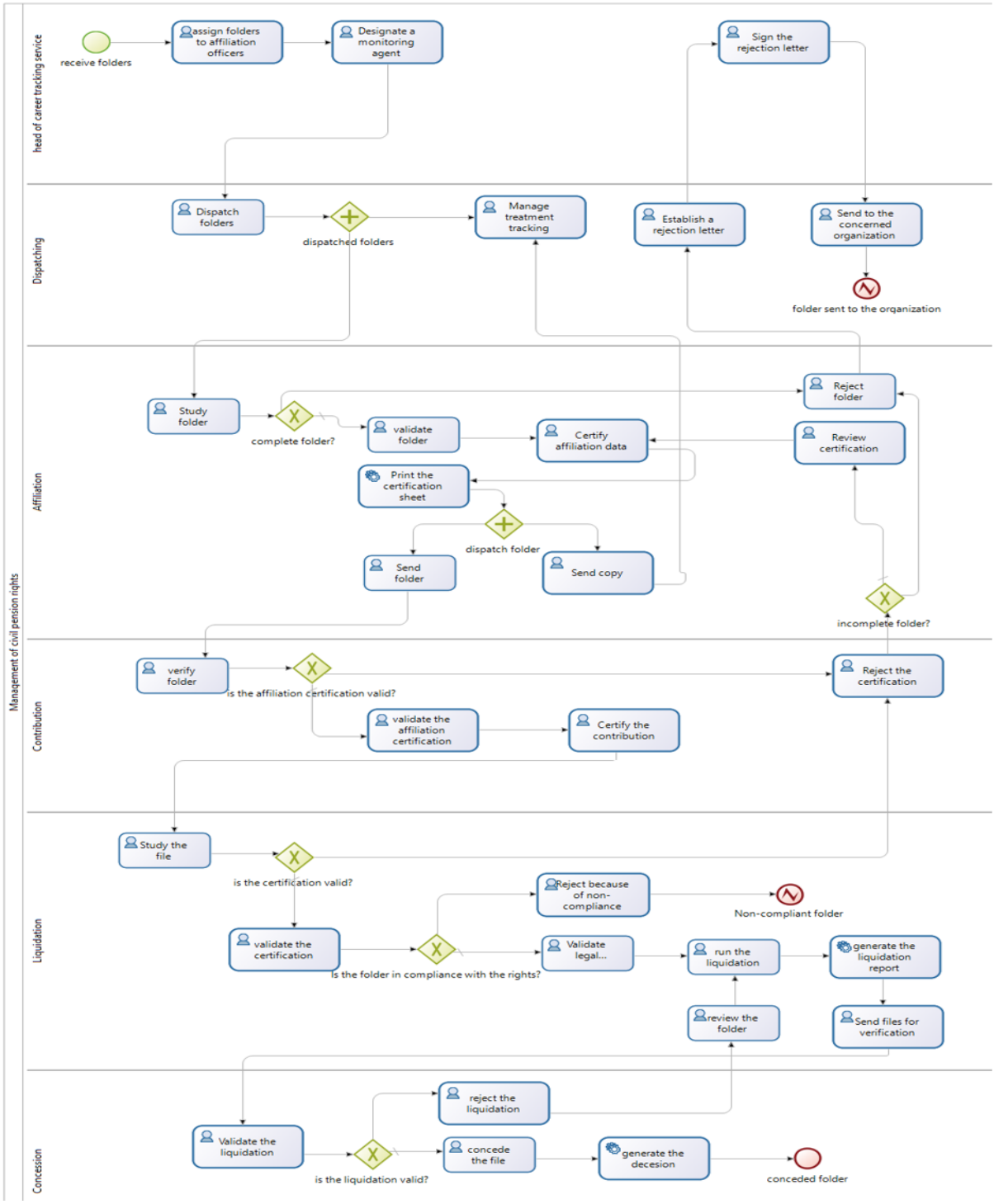

Figure 7 : Le modèle de processus opérationnel de la "gestion des droits de pension civils

L'accent est mis sur l'étude opérationnelle des processus de circulation des dossiers de retraite au sein du MPF, afin de retracer l'itinéraire de chaque dossier depuis sa réception par le service concerné jusqu'à son archivage définit

2. Phase de mesure

La simulation avec Bonita du diagramme précède avec des scénarios : 1000 dossiers des dossiers révisés 5 fois et d'autre sans révision

3. Phase d'analyse

La simulation montre que La date de réception n'affecte pas la durée du traitement, mais elle affecte la date à laquelle le pensionné reçoit sa première pension. Les dossiers qui sont révisé arrivent en retard donc payement en retard et lorsqu'un dossier incomplet est reçu, les activités présentées dans le tableau ne sont pas exécutées, et donc un dossier incomplet n'est pas concédé.

4. Phase d'amélioration

En ce qui concerne le processus opérationnel "gestion des droits de pension civils" en temps réel, ce changement se reflète dans le modèle BPMN par l'ajout de la voie "Contrôle" qui regroupe les activités liées à cette interconnectivité, comme le montre la figure

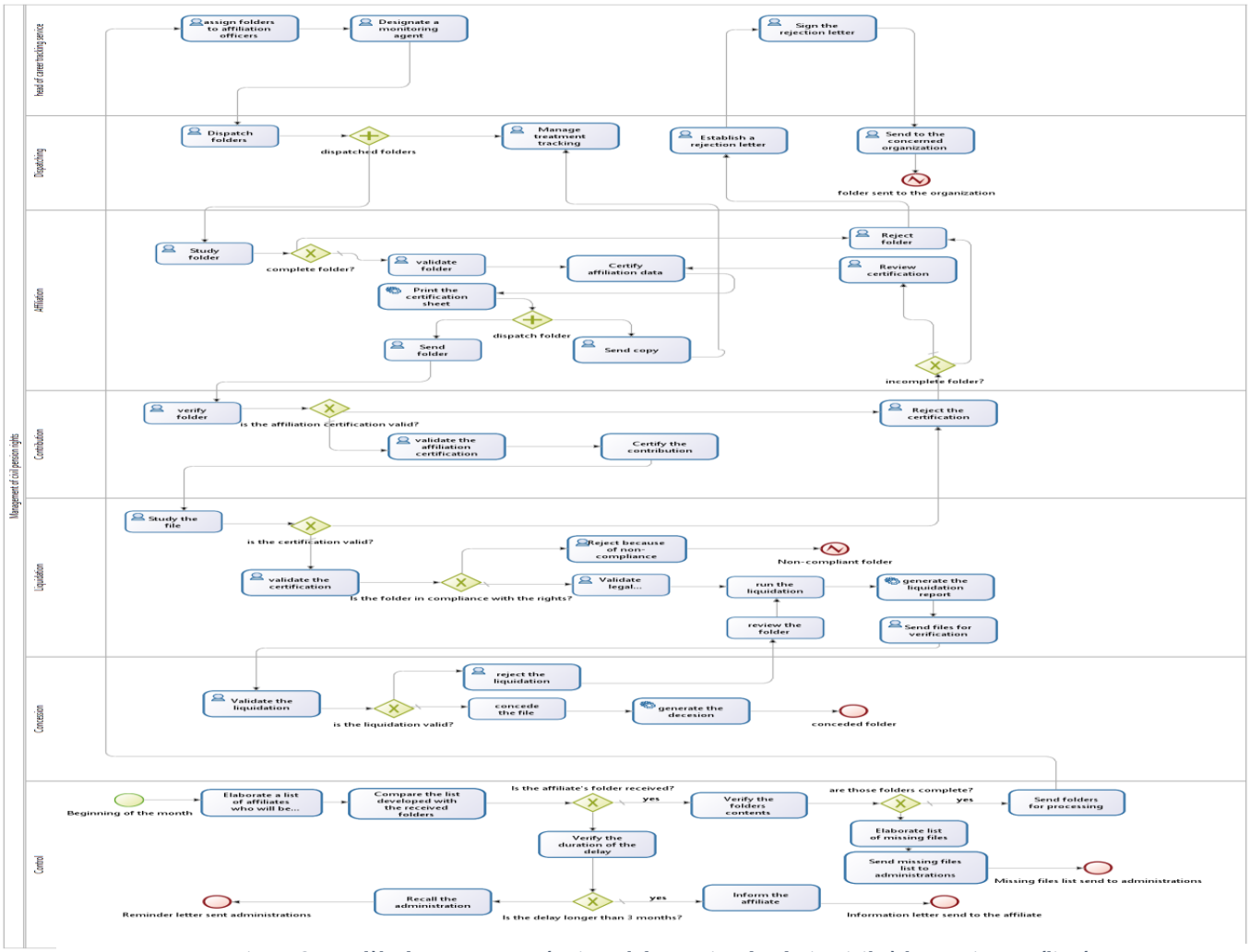

Figure 8 : Modèle de processus opérationnel de "gestion des droits civils à la retraite" amélioré

5. Phase de control

Cette phase nous permet de vérifier continuellement si les actions créées dans la phase précédente sont bien maintenues. Avec des indicateurs de performances ou autre.

Dans notre cas nous allons résoudre les problèmes de circuit des documents par la phase de control mais an aura le problème de la durée d'exécution de processus. Ainsi le système n'est pas auto adaptative. C'est pour cela on va ajouter l'activité 'exécuter un système de recommandation' dans le modèle BPMN pour ce faire nous allons implémenter l'algorithme Q_learning, et obtient les données nécessaires à son exécution à partir de l'activité "Etudier le dossier", renvoyant ainsi la combinaison optimale (état, action) pour le traitement du dossier de pension en cours.

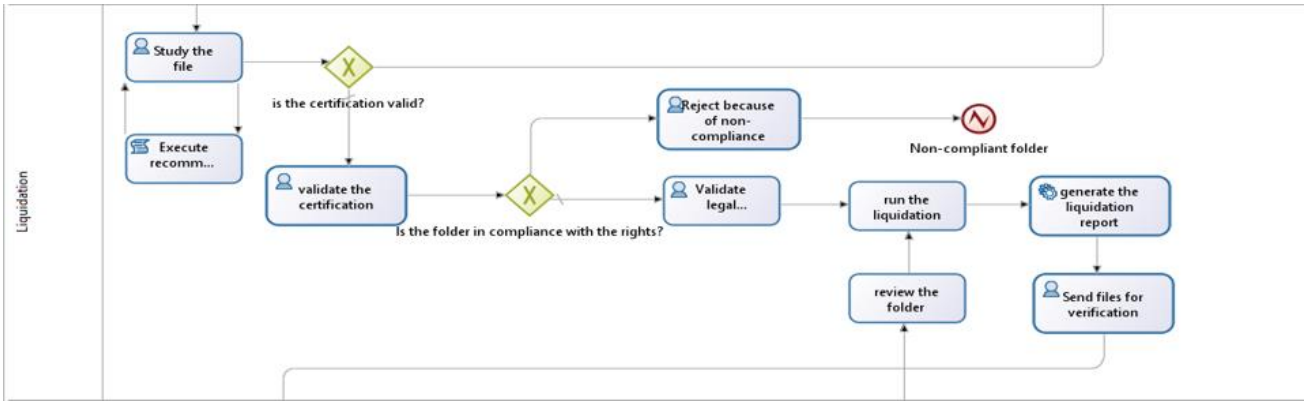

Figure 9 : l'ajout de l'activité 'exécution d'un système de recommandation dans la phase de liquidation du BPMN

Chapitre5

Mise en œuvre du Q_learning et Analyse des résultats

Dans ce chapitre on va présenter l'implémentation du l'algorithme Q_learning pour faire une amélioration de notre système et rendre auto adaptative de chaque état du document de retraite.

I. Outils de développement

Le langage de programmation Python apparu à l'époque en 1991 comme une façon d'automatiser les éléments les plus ennuyeux de l'écriture de scripts ou de réaliser rapidement des prototypes d'applications.

Python est le langage de programmation le plus utilisé dans le domaine du Machine Learning, du Big Data et de la Data Science.

Le principal cas d'usage du Python est le Scripting et l'automatisation. En effet, ce langage permet d'automatiser les interactions avec les navigateurs web ou les GUI d'applications.

Cependant, le Scripting et l'automatisation sont loin d'être les seules utilités de ce langage. Il est aussi utilisé pour la programmation d'applications, pour la création de services web (Le Framework Flask) ou de REST API, ou encore pour la métaprogrammation et pour la génération de code.

II. Implémentation du l'algorithme

1. L'ensemble des états et actions

Pour appliquer l'algorithme Q-learning, la première étape est de définir l'ensemble des états, on vise à obtenir des résultats dans une période limitée. Si l'ensemble est trop grand, l'algorithme prendra beaucoup de temps pour converger. Les étapes par lesquelles passe un dossier de retraite, représente tous les états possibles pour cet agent qui sont : Reçue", "affiliation certifiée", "contribution certifiée", "liquidée", "révisée", "concédée", "rejetée" et "incomplète". Un identifiant est associé à chaque état respectivement comme suit: 'S0: Received', 'S1: Certified affiliation', 'S2: Certified contribution', 'S3: Liquidated', 'S4: Revised', 'S5: Conceded', 'S6: Rejected', 'S7: Incomplete', 'S8: Stuck in a state', 'S9: Returned to a previous state', and 'S10: Skipped a treatment step'. On note S = {S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10} l'ensemble des états.

Les opérations possibles sur le traitement du dossier de retrait présentent toutes les actions de l'agent comme suit : Rejeter", "Certifier l'affiliation du dossier reçu", "Certifier la contribution pour le dossier d'affiliation certifié", "Examiner le dossier de contribution certifié", "Liquider le dossier de contribution certifié", "Concéder le dossier liquidé". Un identifiant est associé à chaque action respectivement comme suit : 'A0: Reject', 'A1: Certify Affiliation of received folder', 'A2: Certify contribution for certified affiliation folder', 'A3: Review certified contribution folder', 'A4: Liquidate certified contribution folder' et 'A5: Concede liquidated folder'. On note A= {A0, A1, A2, A3, A4, A5} l'ensemble des actions.

2. La reward function.

Le but du choix d'une action en transition vers un état est illustré par la reward function qui joue un rôle clé dans la vitesse d'apprentissage. La valeur de la récompense peut être négative ou positive. Une valeur de récompense négative est utilisée pour exprimer une pénalité pour une action indésirable. Les récompenses générées après une transition d'état sont un résultat à court terme. En raison de l'inclusion de futures récompenses dans l'évaluation des actions de l'algorithme, il peut renvoyer une faible récompense immédiate pour une action liée à une Q-value élevée. Jusqu'à maintenant nous avons S l'ensemble des états et A l'ensemble des actions. Les récompenses vont être maintenant donnés à l'agent si un état est accessible depuis un état particulier suivant un action choisie.

	A0	A1	A2	A3	A4	A5
S0	0	10	0			
S1	0	0	10		0	
S2	0	-1	0		10	0
S3			-1	1	0	50
S4				0	10	
S5						
S6						
S7	0					
S8						
S9						
S10						

Figure 10: matrice de récompense incomplète

Prenons un exemple si l'agent est à l'état S0 :'Recieved' et prend l'action A1 :'certifyaffiliation of received folder', alors une récompense positive de 10 vas être affecté puisqu'il passera à l'état suivant S1 :'certified affiliation'. Et s'il prend l'action A0 :'Reject' il passera alors à l'état S0 :'Rejected' et recevra une récompense de 0.

Le but de l'agent est d'arriver à l'état S5 :'Conceded'. Alors, comment pouvons-nous intégrer ce fait dans le tableau ? Cela se fait en associant cet emplacement avec une récompense très supérieure a celle d'habitude. Mettons 50 dans la cellule (S3 :'Liquidated',A5 :'Concede')

Si un état n'est pas directement accessible à partir d'un état particulier suivant une action, nous accordons une récompense de -0.2.

Oui, la récompense n'est qu'un chiffre ici et rien d'autre. Il permet à l'agent de comprendre ses mouvements en lui aidant à décider quels états sont directement accessibles et lesquels ne le sont pas. Avec cette information, nous pouvons construire une table de récompense qui contient toutes les valeurs de récompense pour chaque paire (état,action).

	A0	A1	A2	A3	A4	A5
S0	0	10	0	-0,2	-0,2	-0,2
S1	0	0	10	-0,2	0	-0,2
S2	0	-1	0	-0,2	10	0
S3	-0,2	-0,2	-1	1	0	50
S4	-0,2	-0,2	-0,2	0	10	-0,2
S5	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
S6	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
S7	0	-0,2	-0,2	-0,2	-0,2	-0,2
S8	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
S9	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
S10	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2

Figure 11 : Table des récompenses

Graphe de transition :

Ce figure présente la transition entre les états Si (i=0,1 ,….,10)à partir des action Aj (j=0,1 ,….,5) avec un récompense R.

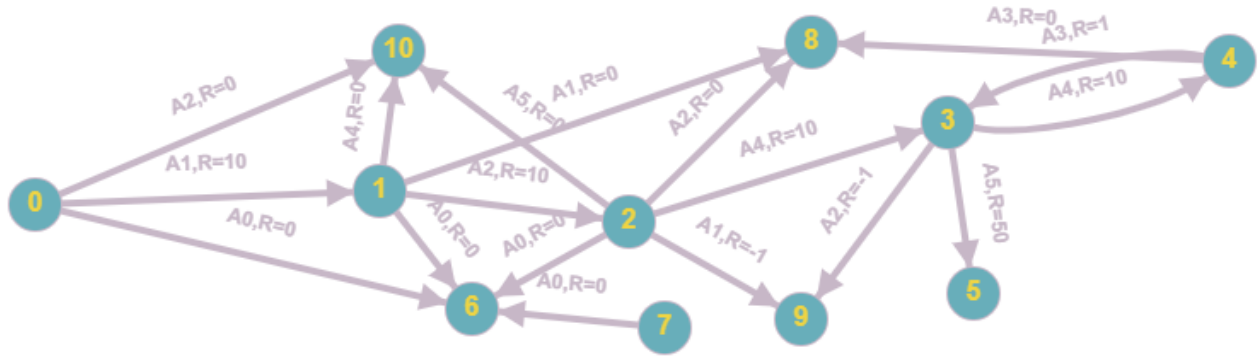

Figure 12 : Graphe de transition

3. La Value function.

Les Q-values des paires état-action sont stockées dans Q-table. Toutes les valeurs stockées dans la Q-table sont initialement égales à zéro. Les lignes du tableau représentent les états du dossier de retrait et les colonnes représentent l'action qui correspond à l'état donné. Pendant l'entrainement, les Q-values sont mises à jour. Après un entraînement suffisant, les Q-values convergent. L'équation de mise à jour est présentée ci-dessous :

$$q(s,a) \leftarrow (1-\alpha)\, q(s,a) + \alpha\,(R_{t+1} + \gamma \max_{a'} q(s',a'))$$

Où :

-'s' est l'état actuel, et 'a' est l'action sélectionnée. 's'' est l'état suivant vers lequel le dossier de retrait est transféré depuis l'état actuel après avoir été exécuté par l'action 'a'.

-'α' est le learning rate qui précise dans quelle mesure les informations récemment acquises remplacent les anciennes informations.

-'γ' est le discount factor, qui determine l'importance des récompenses futures. Après l'entrainement, le dossier de retraite sera capable d'exécuter la meilleure action pour chaque état.

4. Les paramètres du Q-learning

La table de recherche dans laquelle l'agent détermine les récompenses futures maximales anticipées pour une action dans chaque état est appelée Q-table. Cette table conduira l'agent à la meilleure action dans chaque état. Dans la Q-table, les colonnes représentent les actions disponibles et les lignes représentent les états. Chaque résultat Q-table représente la récompense la plus envisagée que l'agent obtiendra s'il exécute cette action à cet état. Afin d'apprendre et d'améliorer chaque valeur de la Q-table à chaque répétition du processus itératif, l'algorithme "Q-learning" est exécuté. Avant d'explorer l'environnement par l'agent, la Q-table accorde des valeurs fixes arbitrairement identiques. Alors que l'agent est encore à l'étape d'exploration, la Q-table attribue une estimation de mieux en mieux en mettant à jour à plusieurs reprises Q (s, a). L'exploration et l'exploitation sont les deux stratégies du processus de formation. L'exploration consiste à trouver plus d'informations sur l'environnement et l'exploitation exploite des informations connues pour maximiser la récompense. L'objectif de l'agent est de maximiser la récompense cumulée attendue. Cela dit, Q-learning utilise ce qu'on appelle la stratégie "epsilon-greedy". Ce taux d'exploration est la probabilité que notre agent explore l'environnement plutôt que de l'exploiter. Au fur et à mesure que l'agent en apprend plus sur

l'environnement, au début de chaque nouvel épisode, ϵ décroîtra à un certain rythme que nous avons fixé de sorte que la probabilité d'exploration devienne de moins en moins probable à mesure que l'agent en apprend de plus en plus sur l'environnement et commence à être plus confiant aux Q-values estimées. Les paramètres utilisés par Q-Learning sont définis dans le tableau (X) comme suit.

Tableau (X) les paramètres deQ-Learning utilisés pour l'agent du dossier de retraite

Parameter	Value
Learning rate	0.05
Discount factor	0.7
Initial Q-value	0
Initial value of ϵ	0.6
ϵ Decay rate	0.005

5. Application de l'algorithme

Pour chaque épisode, l'agent commence pare choisir un état, Pour déterminer si l'agent choisira L'exploration ou l'exploitation à chaque itération, nous générons un nombre aléatoire entre 0 et 1 Supposons que ce nombre est inférieur à ϵ donc l'agent va explorer l'environnement ainsi il va choisir une action aléatoirement. Après avoir choisi l'action, il observe le nouvel état la récompense la récompense obtenue de son action, et met à jour la Q-value dans la Q -table pour l'action qu'il a effectuée à partir de l'état précédent.

Supposons que l'agent est à l'état S0 :'Received' et qu'il a choisi A1 :'Certify affiliation of received folder'. L'état suivant sera alors S1 :'Certified affiliation' et va récupérer sa récompense qui est 10 et doit mettre à jour la la Q-value dans la Q -table en utilisant l'équation de Bellman :

$$q(s,a)=(1-\alpha)\, q(s,a) +\alpha\, (Rt+1+\gamma \max a'\, q(s',a'))$$

$$\mathbf{q(S0,A1) = (1-0.05)q(S0,A1) + \alpha\,(R(S0,A1) + 0.7 * max a' q(S1,a')}$$

$$\mathbf{q(S0,A1) = (1-0.05)q(S0,A1) + 0.05(R(S0,A1) + 0.7 * max a'(q(S1,a' = AO), q(S1,a' = A1), q(S1,a' = A2), q(S1,a' = A3), q(S1,a' = A4), q(S1,a' = A5))}$$

$$\mathbf{= (1-0.05) * 0 + 0.05(10) + 0.7 * max(0,0,0,0,0,0)}$$

$$\mathbf{= 0.5}$$

Puisque les Q-values sont initialisés à 0 dans la Q-table.

Très bien, nous allons maintenant prendre cette nouvelle Q-value que nous venons de calculer et la stocker dans notre Q-table pour cette paire état-action particulière.

Nous avons maintenant fait tout le nécessaire pour une seule itération. Ce même processus se produira pour chaque épisode jusqu'à la fin. Une fois que la fonction Q converge vers la fonction Q optimale, nous aurons notre chemin optimale.

6. Implémentation du Q_learning en Python

Représentant chacun des états avec un nombre. Cela facilitera nos calculs.

```
import numpy as np

states = {
            'Received': 0,
            'Certified affiliation': 1,
            'Certified contribution': 2,
            'Liquidated': 3,
            'Revised': 4,
            'Conceded': 5,
            'Rejected': 6,
            'Incomplete': 7,
            'Stuck in a state': 8,
            'Returned to a previous state': 9,
            'Skipped a treatment step': 10
        }
```

Figure 13 : Dictionnaire de l'ensemble des états

La prochaine étape est de définir les actions.

```
actions = {
             'Reject': 0,
             'Certify Affiliation of received folder': 1,
             'Certify contribution for certified affiliation folder': 2,
             'Review certified contribution folder': 3,
             'Liquidate certified contribution folder': 4,
             'Concede liquidated folder': 5}
```

Figure 14 : Dictionnaire de l'ensemble des actions

Et après la table des récompenses

```
rewards = np.array([[0, 10 ,0 , -0.2, -0.2, -0.2],
                    [0, 0, 10, -0.2, 0, -0.2],
                    [0, -1, 0,-0.2 , 10, 0],
                    [-0.2, -0.2, -1, 1, 0, 50],
                    [-0.2, -0.2, -0.2, 0, 10, -0.2],
                    [-0.2, -0.2, -0.2, -0.2, -0.2, -0.2],
                    [-0.2, -0.2, -0.2, -0.2, -0.2,-0.2],
                    [0, -0.2, -0.2, -0.2, -0.2, -0.2],
                    [-0.2, -0.2, -0.2, -0.2, -0.2, -0.2],
                    [-0.2, 0.2, -0.2, -0.2, -0.2, 0.2],
                    [-0.2, -0.2, -0.2, -0.2, -0.2, -0.2]])
```

Figure 15 : Matrice de récompense en python

Notre prochaine tâche est pour notre agent de se renseigner sur son environnement en mettant en œuvre un modèle de Q-learning. L'ensemble du processus sera répété sur 1000 épisodes. Cela fournira à l'agent une opportunité suffisante d'apprendre les chemins tout en évitant simultanément de s'écraser dans l'un des états avec récompense négative.

Définir les fonctions d'assistance :

Définir une fonction qui détermine si l'état spécifié est un état terminal

```
def is_terminal_state(state):
    if state== 'Conceded' or state=='Rejected' or state=='Stuck in a state' or /
     state=='Returned to a previous state' or state=='Skipped a treatment step':
        return True
    else : return False
```

Figure 16 : fonction qui retourne l'état final du document

Définir une fonction qui choisira un état de départ aléatoire

```
def get_starting_state():
    state=np.random.choice(list(states.keys()))
    return state
```

Figure 17: fonction qui retourne l'état du départ

Définir un algorithme epsilon-greedy qui choisira l'action à entreprendre ensuite. nous définissons un nombre aléatoire compris entre 0 et 1. Cela sera utilisé pour déterminer si notre agent va explorer ou exploiter l'environnement dans cette itération.

```python
def get_next_action(state,epsilon):
    if np.random.random()>epsilon :
        action=np.argmax(Q[states[state],:])
    else:
        action=action=np.random.randint(0,6)
    return action
```

Figure 18 : fonction pour choisir une action

Définir une fonction qui retournera l'état suivant en fonction de l'action choisie pour chaque état choisi

```python
def get_next_state(state,action):
    if state=='Received' and action=='Reject'  : return 'Rejected'
    elif state=='Received' and action=='Certify contribution for certified affiliation folder': return 'Skipped a treatment step'
    elif state=='Received' and action=='Certify Affiliation of received folder' : return 'Certified affiliation'

    elif state=='Certified affiliation' and action=='Reject': return 'Rejected'
    elif state=='Certified affiliation' and action=='Certify contribution for certified affiliation folder': return 'Certified contribution'
    elif state=='Certified affiliation' and action=='Certify Affiliation of received folder' :return 'Stuck in a state'
    elif state=='Certified affiliation' and action=='Liquidate certified contribution folder' :return 'Skipped a treatment step'

    elif state=='Certified contribution' and action=='Reject': return 'Rejected'
    elif state=='Certified contribution' and action=='Certify Affiliation of received folder': return 'Returned to a previous state'
    elif state=='Certified contribution' and action=='Liquidate certified contribution folder': return 'Liquidated'
    elif state=='Certified contribution' and action=='Concede liquidated folder': return 'Skipped a treatment step'
    elif state=='Certified contribution' and action=='Certify contribution for certified affiliation folder' : return 'Stuck in a state'

    elif state=='Liquidated' and action=='Certify contribution for certified affiliation folder' : return 'Returned to a previous state'
    elif state=='Liquidated' and action=='Review certified contribution folder' : return 'Revised'
    elif state=='Liquidated' and action=='Liquidate certified contribution folder' : return 'Stuck in a state'
    elif state=='Liquidated' and action=='Concede liquidated folder' : return 'Conceded'

    elif state=='Revised' and action=='Review certified contribution folder' : return 'Stuck in a state'
    elif state=='Revised' and action=='Liquidate certified contribution folder' : return 'Liquidated'

    elif state=='Incomplete' and action=='Reject' : return 'Rejected'
    else :return 'Received'
```

Figure 19 : fonction pour voir l'état suivant en fonction de l'action choisie pour chaque état choisi

Définir une fonction qui retournera le chemin optimal depuis n'importe quel état lui sera affecté en argument

```python
def get_optimal_route(state):

        starting_state = state
        optimal_route=[]
        optimal_route.append(starting_state)
        while not is_terminal_state(state):
            action=get_next_action(state,0)
            next_state=get_next_state(state,get_action(action))
            optimal_route.append(next_state)
            state=next_state
        return optimal_route
```

Figure 20 : fonction qui retourne le chemin optimal du document

On passe maintenant à la phase d'entrainement de l'agent :

Initialisons les paramètres de l'algorithme

```
#training
epsilon=0.6
discount_factor = 0.7
learning_rate = 0.05
```

Figure 21 : Paramètres du Q_learning

L'agent va parcourir 1000 épisodes d'entraînement, obtenir le lieu de départ de cet épisode, choisir l'action à entreprendre, effectuer l'action choisie et passer à l'état suivant, recevoir la récompense pour le passage au nouvel état et mettre à jour la Q-value pour la paire (état,action)

```
for episode in range(1000):
            state=get_starting_state()
            action=get_next_action(state,epsilon)
            action=get_action(action)
            old_state=state
            next_state=get_next_state(state,action)
            reward=rewards(state,action)

            TD = reward + discount_factor * (np.argmax(Q[states[next_state],])) - Q[states[state],actions[action]]
            Q[states[state],actions[action]] += learning_rate * TD
            print(Q)
```

Figure 22 : mis à jour du Q_table

Maintenant que l'agent a été entièrement formé, nous pouvons voir ce qu'il a appris en affichant le chemin optimal depuis n'importe quel état du dossier de retraite ainsi que la Q-table

```
print(Q)
print('training complete')
print(get_optimal_route('Received'))
```

Figure 23 : code d'affichage du Q_table et le chemin optimal du document

III. Analyse des résultats

Pour chaque épisode, l'agent commence pare choisir un état, Pour déterminer si l'agent choisira L'exploration ou l'exploitation à chaque itération, nous générons un nombre aléatoire entre 0 et 1 Supposons que ce nombre est inférieur à ϵ donc l'agent va explorer l'environnement ainsi il vaChoisir une action aléatoirement. Après avoir choisi l'action, il observe le nouvel état la récompense la récompense obtenue de son action, et met à jour la Q-value dans la Q -table pour l'action qu'il a effectuée à partir de l'état précédent.

1ère itération :

L'agent est à l'état S4 :'Revised' et le nombre aléatoire généré est égale à 0.78 qui est supérieur à ϵ donc il va choisir d'exploiter mais puisque tous les Q-values sont initialement égaux à 0 donc il va choisir aléatoirement une action. L'état suivant sera alors S0 et va récupérer sa récompense qui est -0.2 et doit mettre à jour la Q-value dans la Q -table en utilisant l'équation de Bellman :

$$q(s,a)=(1-\alpha)\, q(s,a) +\alpha\, (Rt+1+\gamma \max a'\, q(s',a'))$$

$$\mathbf{q(S4,A0) = (1-0.05)q(S4,A0) + \alpha\,(R(S4,A0) + 0.7 * maxa'q(S4,a')}$$

$$\begin{aligned}\mathbf{q(S4,A1)} &= \mathbf{(1-0.05)q(S4,A0) + 0.05(R(S4,A0) + 0.7} \\ &\quad \mathbf{* maxa'(q(S4,a'=A0), q(S4,a'=A1), q(S4,a'=A2), q(S4,a'=A3), q(S4,a'=A4), q(S4,a'=A5))} \\ &= \mathbf{(1-0.05) * 0 + 0.05(-0.2) + 0.7 * max(0,0,0,0,0,0)} \\ &= \mathbf{-0.01}\end{aligned}$$

Puisque les Q-values sont initialisés à 0 dans la Q-table.

Très bien, nous allons maintenant prendre cette nouvelle Q-value que nous venons de calculer et la stocker dans notre Q-table pour cette paire état-action particulière.

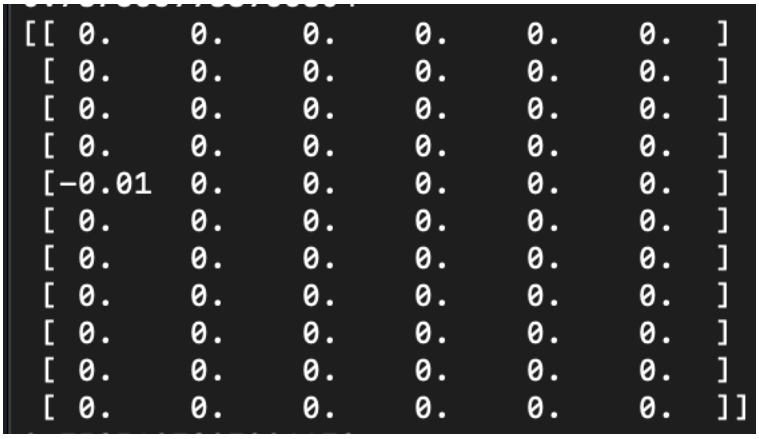

Figure 24 : résultat de la 1ère itération du Q_learning

Itération2 :

l'agent est à l'état S4 :'Revised' et le nombre aléatoire généré est égale à 0.75 qui est supérieur à ϵ donc il va choisir encore une fois d'exploiter mais puisque tous les Q-values sont initialement égaux à 0 sauf pour l'action S0 donc il va choisir aléatoirement une action parmi les 5 dernières.Il a choisi A1 ,l'état suivant sera alors S0 et va récupérer sa récompense qui est -0.2 et doit mettre à jour la Q-value dans la Q -table en utilisant l'équation de Bellman :

$$\mathbf{q(S4,A1) = (1-0.05)q(S4,A1) + \alpha\,(R(S4,A1) + 0.7 * maxa'q(S4,a')}$$

$$\begin{aligned}\mathbf{q(S4,A1)} &= \mathbf{(1-0.05)q(S4,A1) + 0.05(R(S4,A1) + 0.7} \\ &\quad \mathbf{* maxa'(q(S4,a'=A0), q(S4,a'=A1), q(S4,a'=A2), q(S4,a'=A3), q(S4,a'=A4), q(S4,a'=A5))} \\ &= \mathbf{(1-0.05) * 0 + 0.05(-0.2) + 0.7 * max(-0.01,0,0,0,0,0)} \\ &= \mathbf{-0.01}\end{aligned}$$

Nous allons maintenant prendre cette nouvelle Q-value que nous venons de calculer et la stocker dans notre Q-table pour cette paire état-action particulière.

```
[[ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [-0.01 -0.01  0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]]
```

Figure 25 : Résultat de la 2éme itération du Q_learning

Itération 3 :

l'agent est à l'état S2 :' Certified contribution' et le nombre aléatoire généré est égale à 0.15 qui est inférieur à ϵ donc il va choisir cette fois d'explorer l'environnement, du cout il va choisir aléatoirement une action Il a choisi A4 :'Liquidate certified contribution folder', l'état suivant sera alors S3 :'Liquidated et va récupérer sa récompense qui est 10 et doit mettre à jour la

Q-value dans la Q -table en utilisant l'équation de Bellman :

$$\mathbf{q(S2, A4)} = \mathbf{(1 - 0.05)q(S2, A4)} + \boldsymbol{\alpha}\ \mathbf{(R(S2, A4) + 0.7 * maxa'q(S2, a')}$$

$$\begin{aligned}\mathbf{q(S4, A4)} &= \mathbf{(1 - 0.05)q(S2, A4) + 0.05(R(S2, A4) + 0.7} \\ &\quad \mathbf{* maxa'(q(S2, a' = A0), q(S2, a' = A1), q(S2, a' = A2), q(S2, a' = A3), q(S2, a' = A4), q(S2, a' = A5))} \\ &= \mathbf{(1 - 0.05) * 0 + 0.05(10) + 0.7 * max(0, 0, 0, 0, 0, 0)} \\ &= \mathbf{0.5}\end{aligned}$$

On modifie la Q-table

```
[[ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.5    0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [-0.01 -0.01   0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]
 [ 0.     0.     0.     0.     0.     0.  ]]
```

Figure 26 : Résultat de la 3eme itération du Q_learning

Nous avons maintenant fait tout le nécessaire pour les trois premières itérations. Ce même processus se produira pour chaque épisode jusqu'à la fin. Une fois que la fonction Q converge vers la fonction Q optimale, nous aurons notre chemin optimale.

Maintenant que l'agent a été entièrement formé, nous pouvons voir ce qu'il a appris en affichant le chemin optimal depuis n'importe quel état du dossier de retraite ainsi que la Q-table

```
(base) MacBook-Pro-de-mohamed:python mohamedjarni$ python dossierderetraite.py
[[ 1.70727398 18.21539025  2.58930018  2.18678315  1.97843462  1.61046281]
 [ 2.1517798   0.79997437 13.52102703  1.19035011  0.5545486   2.10746669]
 [ 1.43303093  0.19528151  0.52894042 13.38517301  2.23117868  1.70509327]
 [ 0.42718723  0.89067203  1.01550321  0.86197833  2.72488761 49.14115863]
 [ 2.3411044   3.78367009  1.01611429  0.83491265  4.56029168  0.54518763]
 [ 6.08579416  2.23241021  1.17257736  3.28984788  1.36334931  1.19904593]
 [ 1.96893453  1.66315106  1.25119669  1.42062237  4.71598653  1.04821736]
 [ 9.82378921  0.0839043   0.80085375  3.50455654  2.63243907  1.86398117]
 [ 1.37269476  0.86381047  4.36896637  0.90526831  0.9035346   1.31581341]
 [ 0.27607065  2.63900128  1.68082116  1.60666948  1.19431178  0.63342674]
 [ 1.3591108   5.09938306  1.68298389  3.02315721  0.6725251   3.07752762]]
['Received', 'Certified affiliation', 'Certified contribution', 'Liquidated', 'Conceded']
(base) MacBook-Pro-de-mohamed:python mohamedjarni$
```

Figure 27 : Affichage du Q_table et le chemin optimal

Le chemin optimal obtenue si on commence par l'état S0 est ['received', 'certified affiliation', 'certified contribution', 'liquidated', 'conceded'].

Et pour avoir la meilleure action qu'il faut choisir pour chaque état il faut simplement voir la valeur maximale pour chaque action (les colonnes) de cette état (la ligne), cette valeur correspond à l'état optimal. Par exemple si on prend la ligne six c'est-à-dire l'état s6 incomplet ; la valeur maximale se trouve dans l'action de la 1ère colonne est A0 rejeter lequel la meilleur action.

Conclusion

Au terme de ce projet, nous avons réussi l'objectif principal que nous nous sommes fixés dans le premier chapitre, à savoir l'implémentation de l'algorithme Q-learning au dossier de retraite.

Pour ce faire, nous avons commencé tout d'abord par une description générale du projet pour identifier la problématique Concernant le processus opérationnel étudié, qui est en l'occurrence "la gestion des droits à pension civils" du SC de retraite auto-adaptative présenté dans MPF (la caisse de retraite marocaine).Après nous avons présenté l'évolution de l'algorithme du Q_learning et sa relation avec le processus de décision de Markov ainsi nous avons déterminé les variances du Q_learning pour justifier le choix de notre modèle. Après, nous avons passé à l'exploration de la chaine logistique de retraite par les outils de modélisation **SCOR** et **BPMN** en utilisant la modèle DMAIC de Six Sigma pour faire l'évaluation de notre Processus. Enfin nous avons adopté l'algorithme Q_learning pour faire une amélioration de notre système et rendre auto adaptative de chaque état du document de retraite.

Ce projet nous a été bénéfique sur plusieurs plans. Il nous a permis de découvrir l'apprentissage par renforcement et se familiariser avec ses notions et sa relation avec le processus de décision de Markov, maîtriser les méthodes, concepts et outils du développement avec python, de consolider notre formation théorique et pratique, la modélisation SCOR et BPMN et de nous familiariser avec les outils acquis durant toute l'année.

En termes de perspectives, nous souhaitons finaliser développer un service web pour améliorer la supply chain des dossiers de retraite par l'apprentissage par renforcement en appliquant l'algorithme Q_learning et faciliter son utilisation.

Bibliographie

https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf