Final project : Load Balancing

Lahmouz Zakaria Karmaoui Oussama

December 3, 2023

1 Introduction

Time is discrete. Let Q_1 and Q_2 denote the total number of jobs in the server 1 and 2, respectively. Every time slot, the dispatcher observes (Q_1, Q_2) and takes the action a_i, $i = 1, 2$, that dispatches a potential new incoming job to server i. The cost in every time slot is $Q_1 + Q_2$, independently of the action.

In every time slot, there is a probability λ of having a new job, which will be dispatched to server a_i. If $Q_i > 0$, with probability μ_i a job from server i will depart. For simplicity, we will assume that in every time slot, only one event can happen, i.e., either an arrival, or a departure from servers 1 or 2.

For example, in the state $(2, 1)$, if the dispatcher takes the action 1, we have that with probability λ the next state will be $(3, 1)$, with probability $\mu_1(1, 1)$, with probability $\mu_2(2, 0)$, and with probability $1 - \mu_1 - \mu_2 - \lambda$ the next state will be $(2, 1)$.

We will assume that there is an upper bound for both Q_1 and Q_2 equal to 20. If either $Q_1 = 20$ or $Q_2 = 20$, no new incoming job will arrive to the system.

Let us choose $\mu_1 = 0.2$, $\mu_2 = 0.4$ and $\lambda = 0.3$. Throughout we take that the discouting factor is $\gamma = 0.99$.

2 MDP

2.1 Policy evaluation

2.1.1 L'équation de Bellman

L'équation de Bellman qui caractérise le problème s'écrit sous la forme : $V(s) = r(s) + \gamma \sum_{s'} p(s', s) * V(s')$.
Les parametres d'entrée de la fonction de Bellman sont :
r(s) = E(R_{t+1} | s) espérance de récompenses futures sachant l'état courante s Q1: le nombre de jobs dans le serveur 1,
Q2: le nombre de serveurs dans le serveur 2,
γ: le facteur de discount,
λ: la probabilité pour qu'un job sera transmis à l'un des serveurs,
μ_1; la probabilité de sortir d'un job du serveur 1 et μ_2 la probabilité de sortir d'un job du serveur2.

On choisit une politique aléatoire, c'est a dire il y a équiprobabilité de distrubition d'un job sur le serveur 1 ou le serveur 2. Ainsi pour les valeurs normales (internes) de $Q1$ et $Q2$:
$V(s) = 0.5 * \lambda * (-2(Q1 + Q2 + 1) + \gamma * (V(Q1 + 1, Q2) + V(Q1, Q2 + 1) + \mu_1(-(Q1 + Q2 - 1) + \gamma V(Q1 - 1, Q2) + \mu_2 * (-(Q1 + Q2 - 1) + \gamma * V(Q1, Q2 - 1)) + (1 - \lambda - \mu_1 - \mu_2) * (-(Q1 + Q2) + \gamma * V(Q1, Q2)))$.
Dans le code on a traiter aussi les valeurs limites de Q1 et Q2 (0 ou 20) , ou les probabilités P(S'| S=(Q1,Q2)) sont mise à jour.

La figure 1 représente la matrice value iteration pour une politique aléatoire uniforme.

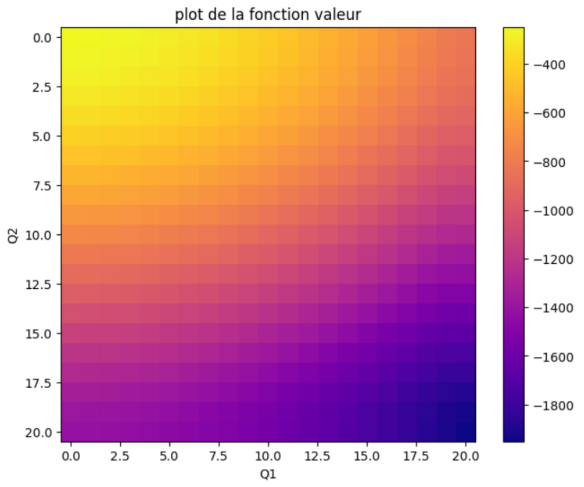

Figure 1: Représentation de la matrice value pour une politique aléatoire (1/2 pour a1 et a2)

Remarque : On observe une moyene de valeurs entre -600 et -1200 pour les valeurs intermédiaires de Q1 et Q2 , les valeurs maximales correspondent au cas de (Q1,Q2) = (0,0) (à peu près -400) , et les valeurs minimales correspondent au cas où les serveur sont saturées (valeurs environ -1800 pour Q1 et Q2 = 20).

2.2 Bonus : Bellman Equation en forme matricielle

D'autre part,on pourra resoudre le probleme par le calcul direct de la matrice V .En effet V s'écrit sous la forme $V = (I - \gamma P)^{-1} R$ ou P la matrice des probabilités et R la matrice des rewards. Les deux approches donnent les résultats identiques à 10-5 près (partie code).

2.3 Optimal Control

2.3.1 Equation de Bellman

L'équation de Bellman qui caractérise la politique optimale (notée π^*) s'écrit sous la forme : $V^*(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s') \right)$, ou $V^* = V_{\pi *}$ la fonction valeur associée à cette politique optimale .

2.3.2 Résolution de la fonction value optimale par l'algorithme value iteration

En initialisant V avec matrice nulle , on fait une mise à jour à la matrice value function en utilisant l'équation de bellman d'optimalité appliquée sur une matrice précedente V_{prev} jusqu'à atteindre une précision définie entre V et V_{prev} (à peu près 10^-6) , mais cette fois en calcul de meme l'action optimale pour chaque état en utilisant l'équation bellman , ce qui donne les 2 figures ci-dessous :

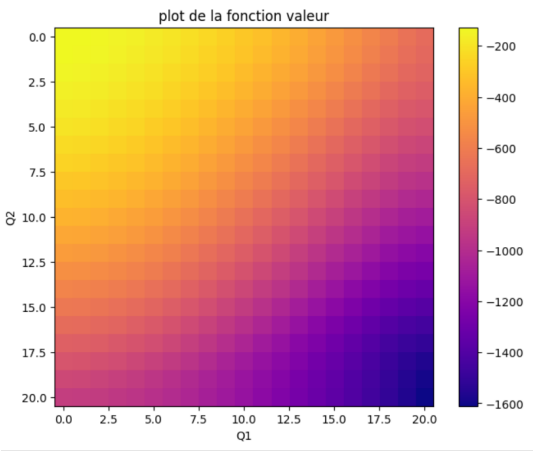

Figure 2: Matrice de la fonction de valeur optimale

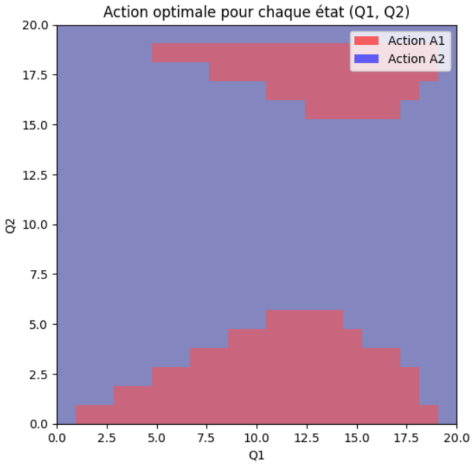

Figure 3: Graphe représentant l'action optimale selon l'état (Q1,Q2)

Remarque : Lorsque Q2 est entre 5 et 15 , l'action optimale pour augmenter les rewards est l'action a2 , ce qui est normale puisque la probabilité de la sortie d'une tache de serveur 2 $mu_2 = 0.4$ est plus grande que celle de serveur 1 $mu_1 = 0.2$, donc probablement le cout qui est la somme de Q1 et Q2 va diminuer avec le temps puisque Q2 n'atteint pas encore ses limites (0 ou 20). Or , lorsque les valeurs de Q2 sont à l'extreme (0 ou 20), c'est plus utilise de choisir l'action a1 parsqu'il est plus probable que Q2 arrive à 0 (pas de tache sortie alors) ou de 20 (système se bloque , aucune tache à introduire.

2.3.3 Comparaison de performances entre méthode aléatoire et controle optimale :

Selon les figures 1 et 2 , les valeurs de la value function obtenue par optimal control sont plus grande que celle obtenue par la première méhode (une moyenne de -700 et des valeurs entre -1600 et -127 , en comparant avec la value function de random policy obtenue ayant une moyenne de -900 et des valeurs qui arrivent jusqu'à -1950) , ce qui montre la performance de la méthode de controle optimale.

2.3.4 Changer la loi aléatoire de la politique

Dans cette partie on a décidé de changer la loi policy aleatoire uniforme de distribution des jobs aux serveurs 1 et 2 par la loiBernoulli de paramètre p.
En comparaison avec la figure 1,les figure 4 et 5 montrent que plus la valeur de (p-0.5) augmente (c'est à dire la probabilité que tous les jobs sont transmis à un unique serveur est grande) plus la probabilité d'un cout élevé augmente et donc plus le reward diminue.

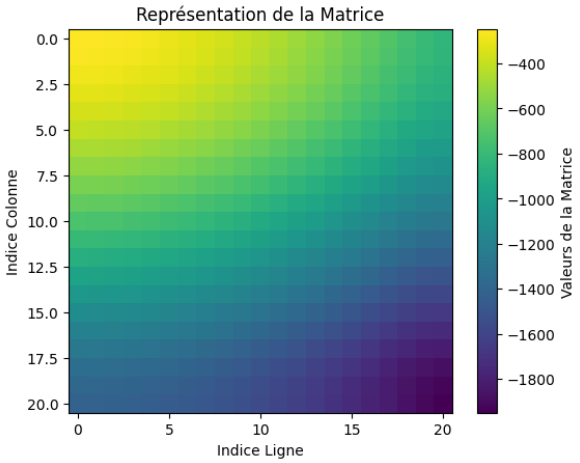

Figure 4: La matrice Value fonction pour p=0.5 probabilité qu'un job est transmis au serveur 1 (politique à loi Bernoulli) .

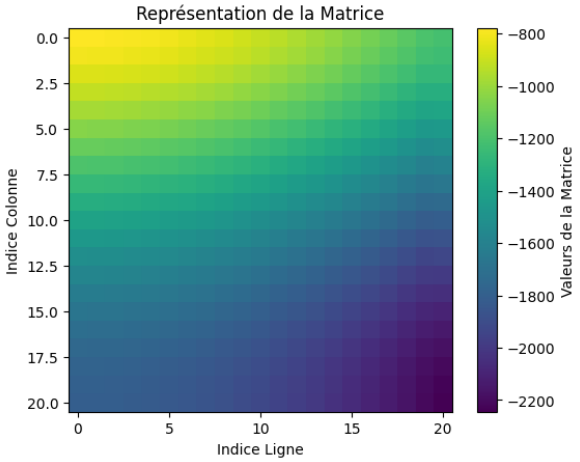

Figure 5: La matrice Value fonction pour p=0.9 probabilité qu'un job est transmis au serveur 1 (politique à loi Bernoulli).

3 Tabular Model-Free control

3.1 Policy evaluation

3.1.1 TD(0) algorithm

Dans cette partie, nous utilisons une politique aléatoire qui répartit chaque tâche avec une probabilité de 0.5 vers le serveur i (i € 1,2). Une action aléatoire est choisie avec une probabilité égale. Nous utilisons la règle de mise à jour TD(0) pour mettre à jour la fonction de valeur pour chaque état , en estimant les états prochaines (s') à parir des probabilités de l'état prochaine sachant l'état courante s (simulation d'une variable aléatoire uniforme sur [0,1] et la comparer avec ces probas) , de meme ,les récompenses futures actualisées et la différence entre l'estimation actuelle et l'estimation mise à jour sont prises en compte : V(s) = V(s) + alpha(-r(s') + gamma*V(s') - V(s)). Pour le paramètre d'apprentissage, nous choisissons ici alpha n = 1/n avec n comme numéro d'épisode et on a utilisé au début 500 épisodes de répétition.Nous obtenons la fonction de valeur et la politique optimale ci-dessous :

Remarque1 : La fonction de valeur obtenue avec cet algorithme est plus grande (valeurs entre -200 et -100) à celle obtenue dans la partie 1, comme nous pouvons le constater.

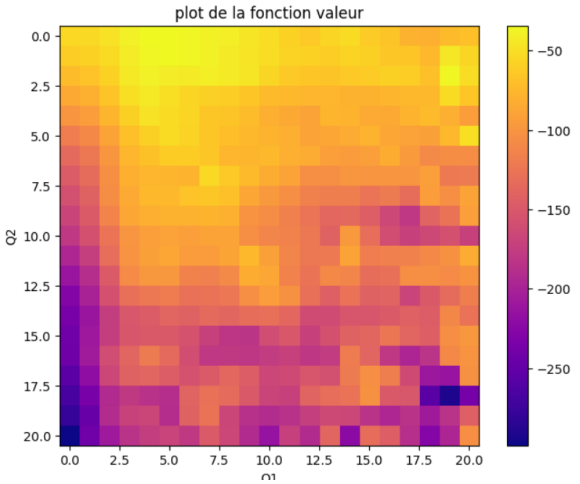

Figure 6: Représentation de la matrice value function en utilisant l'algo TD(0) avec $\alpha_n = 1/n$.

Si on choisit une valeur constante pour la suite α_n , par exemple 1/2 :

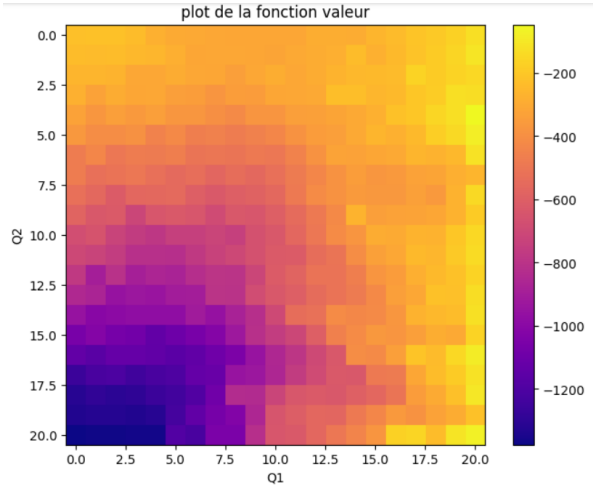

Figure 7: Représentation de la matrice value function en utilisant l'algo TD(0) avec $\alpha_n = 1/2$.

On constante que les valeurs sont très petites en comparant avec celle en utilisant $\alpha_n = 1/\text{n}$, mais d'un autre coté , la norme entre V_{n+1} et V_n (ou V et V_{prev} dans l'itération n de TD(0)) devient plus petite ce qui accèlère le temps d'execution , de meme , la répartition de valeurs comme on voit sur la figure 6 devient plus claire, améliorant ainsi la convergence mais avec moins de performance au coté de valeurs calcuées (valeurs plus petites).

3.2 Optimal control

3.2.1 L'algorithme Q learning

Dans cette partie on utiise l'algorithme QLearning pour trouver la policy optimale.La figure ci dessous représente les étapes fondamentales de l'algorithme:

l'algorithme Qlearning se caractérise par maximiser la fonction Q pour chaque état et chaque action.Le processus d'apprentissage commence par l'initialisation arbitraire de la matrice Q.Ensuite, l'agent choisit une action pour l'état actuel en utilisant une stratégie d'exploration-exploitation. en particulier epsilon-greedy policy, où l'agent choisit l'action optimale avec probabilité $1-\epsilon$ et une action aléatoire avec probabilité ϵ.Puis,l'agent prend l'action choisie, observe la récompense de l'environnement et passe à l'état suivant.Aprés, on fait la mise à jour de la matrice Q suivant la formule $Q(s,a) = Q(s,a) + \alpha(r(s') + \gamma * \max(Q(s',a')) - Q(s,a))$puis on répète les étapes précédentes jusqu'à la convergence de Q.

3.3 Resultats numériques de l'action optimale

la figure 8 représente la matrice action optimale en utilisant l'algorithme Q-learning.

A propos du choix du learning rate $\alpha = \frac{1}{n^\gamma}$,on constate que plus on augmente γ plus la converegence est plus rapide mais le reward diminue en augmentant γ

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

Figure 8: QLearning algorithm

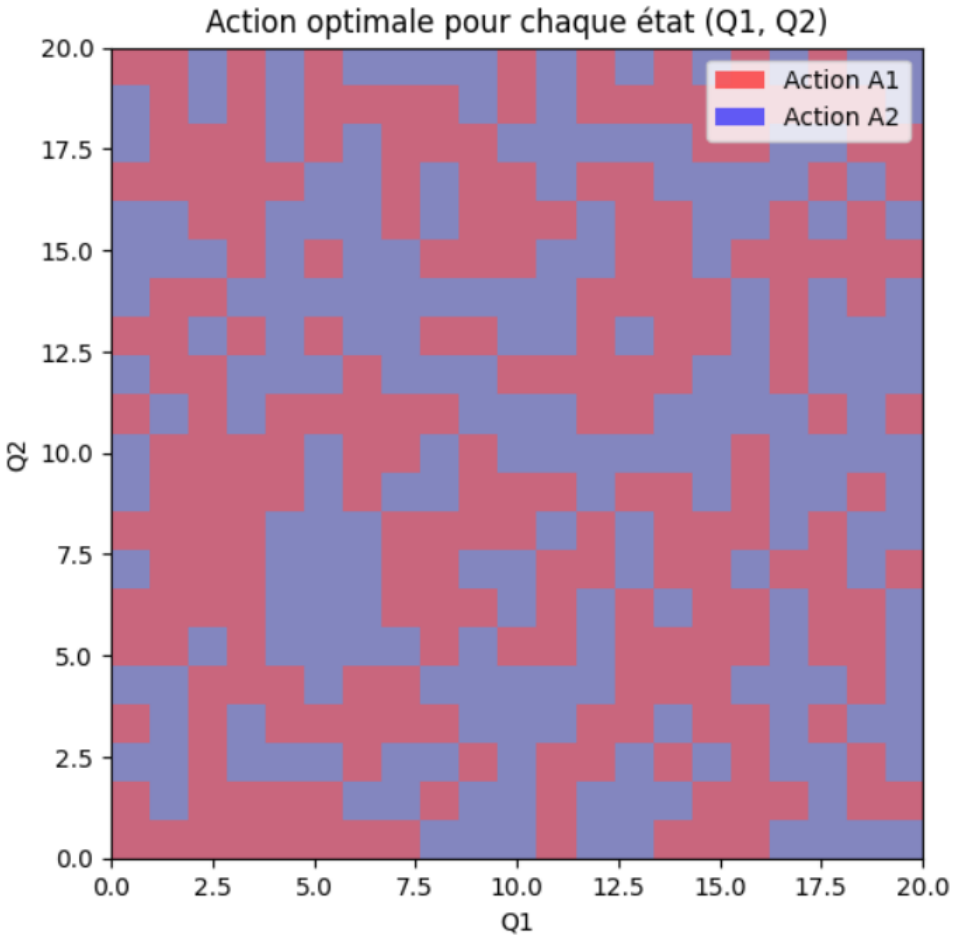

Figure 9: Représentation de la matrice action.