Université Libre de Bruxelles

INFO-F311
Intelligence Artificielle

Synthèse IA

Étudiants :
Rayan Contuliano Bravo

Enseignants :
T. Lenaerts

3 janvier 2024

Contents

1 Markovian Decision Processes

Définition 1.1. Markovian Decision Processes

Un **Markovian Decision Process** (MDP) est un modèle de processus de décision séquentiel dans lequel les états sont Markoviens. Un MDP est défini par un tuple $\langle S, A, T, R, \gamma \rangle$ où :

- S est un ensemble fini d'états ;
- A est un ensemble fini d'actions ;
- $T : S \times A \times S \to [0, 1]$ est la fonction de transition. On a que $T(s, a, s') = P(s'|s, a)$ qui est la probabilité de passer de l'état s à l'état s' en effectuant l'action a ;
- $R : S \times A \times S \to \mathbb{R}$ est la fonction de récompense ;
- S_0 est l'état initial ;
- S_T est l'état terminal ;

Nous sommes donc dans un environnement **Non-Déterministe**, **Séquentiel**, **Statique**, **Discret** et **Complètement observable**. Une action a n'a pas toujours le même effet dans un état s donné. En effet, il y a une probabilité $T(s, a, s')$ que l'action a dans l'état s nous amène à l'état s'.

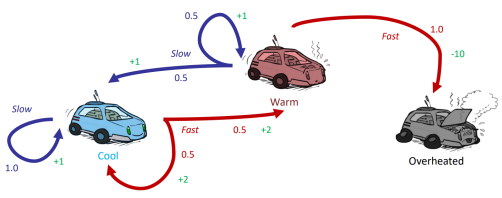
Dans une recherche déterministe, nous pouvons toujours déterminer l'état suivant. Par ce fait nous voulions établir un **plan** qui est une séquence d'actions qui nous amène à l'état final. Dans un MDP, nous ne pouvons pas établir un plan car nous ne pouvons pas déterminer l'état suivant à 100% de certitude. Nous devons donc établir une **politique** qui est une fonction qui nous dit quelle action prendre dans un état donné.

Définition 1.2. Politique

Une **politique** est une fonction $\pi : S \to A$ qui nous dit quelle action prendre dans un état donné. La politique **optimale** est la politique qui maximise l'utilité, notée π^*.

Remarque: Un plan n'est pas une bonne idée dans un environnement non-déterministe car il peut y avoir des changements dans l'environnement qui rendent le plan **obsolète**.

Exemple: Imaginons une voiture qui peut se déplacer soit rapidement, soit lentement. En fonction de ses actions, elle est soite *Cool*, *Warm* ou *Overheated*. Voici la représentation via un automate :



s	a	s'	T(s,a,s')	R(s,a,s')
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	–10
	(end)		1.0	0

FIGURE 1.1 – Voiture de course (kachow)

Nous pouvons alors représenter l'arbre de recherche de la voiture de course comme suit :

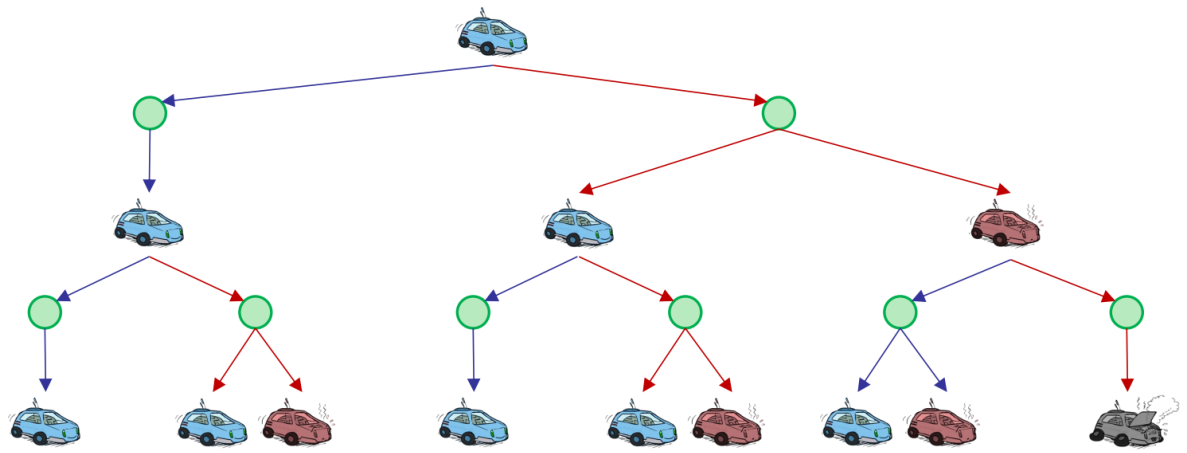


FIGURE 1.2 – Arbre de recherche

Nous pouvons faire un lien entre la représentation de l'arbre et l'arbre de l'algorithme **Expectimax** vu précédemment. En effet, nous pouvons voir que les nœuds chance sont les nœuds où la voiture peut changer d'état en fonction d'une action.

1.1 Hyper-Paramètres

Définition 1.3. Discount Factor

Le **discount factor** γ est un hyper-paramètre qui représente l'importance des récompenses futures. Plus γ est proche de 1 ou supérieur, plus les récompenses futures sont importantes. Plus γ est proche de 0, plus les récompenses futures sont négligeables.

- $\gamma = 0$: **Myopic** (court-termiste) ;
- $\gamma = 1+$: **Far-sighted** (long-termiste) ;

Si $\gamma = 1$, les récompenses futures sont aussi importantes que les récompenses actuelles.
A chaque étape de la recherche, nous multiplions la récompense par γ^k où k est le nombre d'étapes

Définition 1.4. Living Reward

Le **living reward** est une récompense négative qui est donnée à chaque étape de la recherche. Cela permet d'éviter que l'agent ne reste dans un état pendant trop longtemps.

Définition 1.5. Noise

Le **noise** est un hyper-paramètre qui représente la probabilité que l'action choisie ne soit pas celle effectuée. Cela permet de représenter le fait que l'agent ne peut pas toujours effectuer l'action qu'il veut. Par exemple, si l'agent veut aller à gauche, il peut y avoir un obstacle qui l'en empêche et va donc aller à droite. Le noise permet de représenter ce genre de situation.

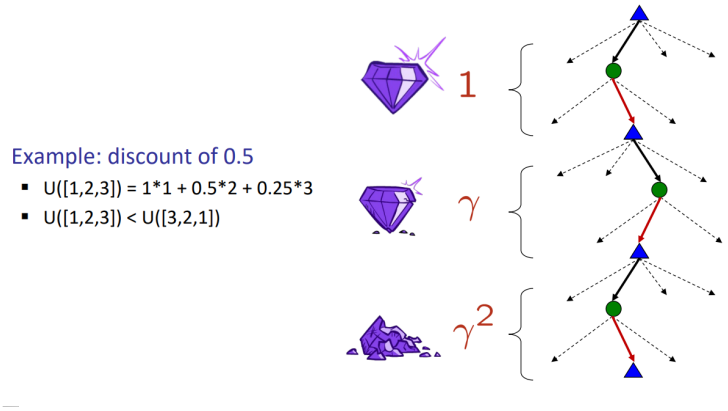


FIGURE 1.3 – Illustration du discount factor. $U()$ = l'utilité = $Reward \times \gamma^k$

L'utilité d'une politique π est définie comme suit :

$$U^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k, s_{k+1})\right] \tag{1}$$

C'est-à-dire que l'utilité d'une politique est la somme des récompenses futures pondérées par le discount factor.

1.2 Equation de Bellman

1.2.1 Quantité optimale

Définition 1.6. Utilité d'un état

Nous définissions l'utilité d'un état s comme la valeur $V*(s)$. C'est l'utilité espérée à partir de s en suivant la politique optimale π^*.

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

Définition 1.7. Utilité d'un q-état

Nous définissions l'utilité d'un q-état s, a comme la valeur $Q*(s, a)$. C'est l'utilité espérée à partir de s en suivant la politique optimale π^* et en effectuant l'action a.

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right]$$

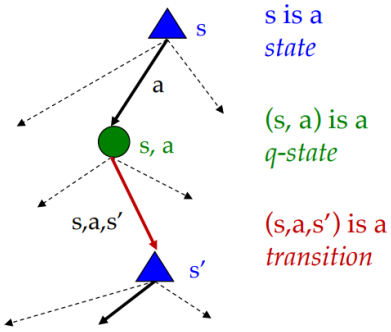


FIGURE 1.4

1.2.2 Value Iteration

Algorithm 1 Value Iteration

procedure ValueIteration($S, A, T, R, \gamma, \epsilon$)
  $V(s) \leftarrow 0$ for all $s \in S$
  repeat
    $V_k \leftarrow V$ Copie de V
    for all $s \in S$ **do**
      $V_k(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$
    end for
    $V \leftarrow V_k$ Mise à jour de V
  until $\max_{s \in S} |V(s) - V'(s)| < \epsilon$
  return V
end procedure

L'algorithme est répété jusqu'à ce que la différence entre V et V' soit inférieure à ϵ. Soit ce que l'on appelle la **convergence**. L'algorithme va converger au bout d'un certain nombre d'itérations. En effet, plus le nombre d'itérations augmente, plus la différence entre V et V' diminue de part le discount factor γ qui devient de plus en plus petit.

Complexité : $O(|S|^2 \times |A|)$

1.2.3 Policy Extraction

Algorithm 2 Policy Extraction

procedure PolicyExtraction(S, A, T, R, γ, V)
  $\pi(s) \leftarrow_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$
  return π
end procedure

Cet algorithme permet d'extraire la politique optimale π^* à partir de V^*. En effet, nous pouvons voir que la politique optimale est la politique qui maximise l'utilité. Nous pouvons donc extraire la politique optimale à partir de V^* en choisissant l'action qui maximise l'utilité pour chaque état.

1.2.4 Policy Evaluation

Cet algorithme permet de calculer l'utilité d'une politique π donnée. Il va calculer l'utilité de chaque état en fonction de la politique π et de la fonction de transition T. Il y a moins de calculs à faire que pour l'algorithme de **Value Iteration** car nous n'avons pas besoin de calculer toutes les valeurs de V à chaque itération. Nous avons juste besoin l'utilité d'un état en fonction **d'une seule** action.

Algorithm 3 Policy Evaluation

procedure PolicyEvaluation(S, A, T, R, γ, π)
  $V(s) \leftarrow 0$ for all $s \in S$
  repeat
    $V_k \leftarrow V$ Copie de V
    for all $s \in S$ **do**
      $V_k(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V(s')\right]$
    end for
    $V \leftarrow V_k$ Mise à jour de V
  until $\max_{s \in S} |V(s) - V'(s)| < \epsilon$
  return V
end procedure

Nous pouvons voir que l'algorithme est similaire à l'algorithme de **Value Iteration** mais que nous n'avons pas besoin de calculer la somme des utilités de toutes les actions possibles. Nous avons juste besoin de calculer l'utilité d'une seule action. Cela permet de réduire le nombre de calculs à faire.

Complexité : $O(|S|^2)$

1.2.5 Policy Iteration

L'algorithme de **Policy Iteration** est un algorithme qui permet de trouver la politique optimale π^* à partir d'une politique initiale π_0. Il va alterner entre les étapes de **Policy Evaluation** et **Policy Extraction** jusqu'à ce que la politique ne change plus. La différence avec l'algorithme de **Value Iteration** est que on commence avec une politique initiale π_0 et on va la modifier jusqu'à ce qu'elle soit optimale. On a donc moins de calculs à faire car on ne va pas devoir calculer toutes les valeurs de V à chaque itération.

Algorithm 4 Policy Iteration

procedure PolicyIteration($S, A, T, R, \gamma, \epsilon$)
  $\pi(s) \leftarrow$ random policy
  repeat
    $V_{\pi k+1} \leftarrow$ PolicyEvaluation(S, A, T, R, γ, π)
    $\pi_{i+1} \leftarrow$ PolicyExtraction(S, A, T, R, γ, V)
  until π does not change
  return π
end procedure

Complexité : $O(|S|^2 \times |A|) + O(|S|^3)$

Cette algorithme est plus lent mais converge plus rapidement que l'algorithme de **Value Iteration** *(moins d'itérations)*.

Remarque: Voir exemple slide 22 cours 15.