Approche par estimation directe

- Cette approche est similaire à l'apprentissage supervisé
  - on apprend à partir de paires
    (état visité *s*, somme pondérée des récompenses à partir de *s*)

- L'estimation **ignore la relation récursive entre les valeurs**, décrite par les équations de la fonction de valeur

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s))\, V(s')$$

- Par exemple, bien que l'essai 1 dit rien sur (3,2), elle nous apprend que (3,3) a une valeur élevée

- On pourrait également déduire que (3,2) a une valeur élevée, puisque (3,2) est adjacent à (3,3)

Hugo Larochelle et Froduald Kabanza

Approche par programmation dynamique adaptative

- C'est l'idée derrière la **programmation dynamique adaptative** (PDA)
  - tirer profit des équations de la fonction de valeur pour estimer $V(s)$

- L'approche par PDA **n'apprend pas directement $V(s)$**, mais apprend plutôt le modèle de transition $P(s'|s, a)$
  - étant donnée une estimation de $P(s'|s, a)$, on peut résoudre $V(s) = R(s) + \gamma \ \Sigma_{s' \in S} P(s'|s, \pi(s)) \ V(s')$
  - on obtient alors notre estimation de $V(s)$
- On peut estimer $P(s'|s, a)$ à partir des fréquences des transitions observées:

$$P(s'|s, \pi(s)) = \frac{\sum_{\text{essais}} \text{freq}(s',s)}{\sum_{\text{essais}} \text{freq}(s)}$$

nb. de transitions de s à s' dans l'essai

nb. de fois que s est visité dans l'essai

somme sur tous les essais

Approche par programmation dynamique adaptative

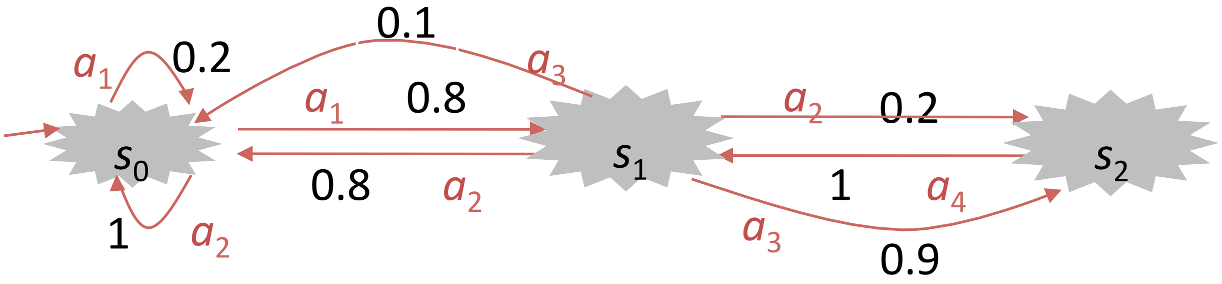
```
function PASSIVE-ADP-AGENT(percept) returns an action
   inputs: percept, a percept indicating the current state s' and reward signal r'
   persistent: π, a fixed policy
               mdp, an MDP with model P, rewards R, discount γ
               U, a table of utilities, initially empty
               N_sa, a table of frequencies for state–action pairs, initially zero
               N_s'|sa, a table of outcome frequencies given state–action pairs, initially zero
               s, a, the previous state and action, initially null

   if s is not null then
       increment N_sa[s, a] and N_s'|sa[s', s, a]
       for each t such that N_s'|sa[t, s, a] is nonzero do
           P(t | s, a) ← N_s'|sa[t, s, a] / N_sa[s, a]
   U ← POLICY-EVALUATION(π, U, mdp)
   if s'.TERMINAL? then s, a ← null else s, a ← s', π[s']
   return a
```

Fonction qui résout

$V(s) = R(s) + \gamma \ \Sigma_{s' \in S} P(s'|s, \pi(s)) \ V(s')$

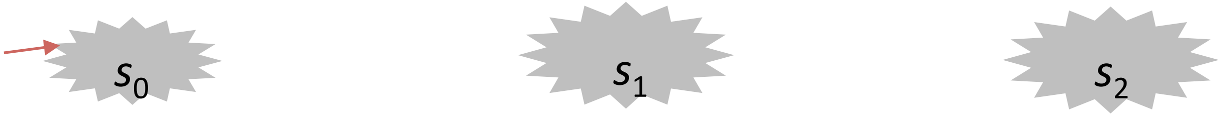
Approche par programmation dynamique adaptative

- Exemple (**avec état terminal**)



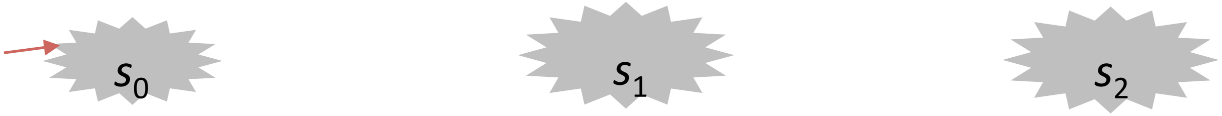
  - MDP à 3 états: $S = \{s_0, s_1, s_2\}$
  - fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
  - le facteur d'escompte est $\gamma=0.5$
  - s_2 est un état terminal, s_0 est l'état initial
  - plan suivi: $\pi(s_0) = a_1$, $\pi(s_1) = a_3$

Approche par programmation dynamique adaptative



- Initialement, on suppose aucune connection entre les états

Approche par programmation dynamique adaptative

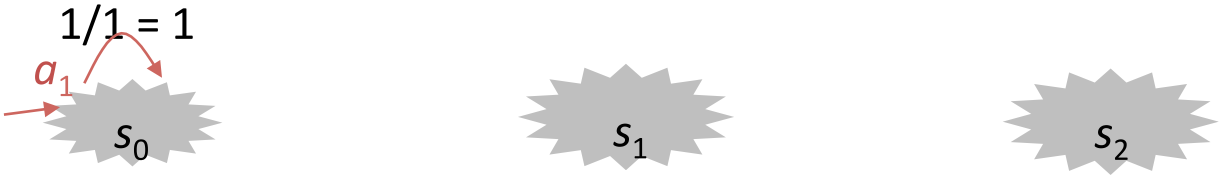


- Observations: (s_0) $_{-0.1}$

  $V(s_0) = -0.1$
  $V(s_1) = -0.1$
  $V(s_2) = 1$

Hugo Larochelle et Froduald Kabanza

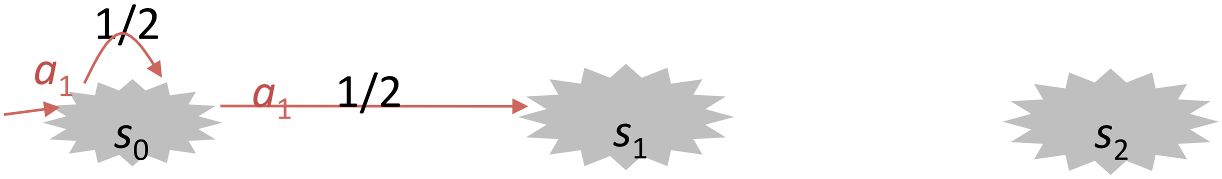
Approche par programmation dynamique adaptative



- Observations: $(s_0)_{\text{-0.1}} \rightarrow (s_0)_{\text{-0.1}}$

$V(s_0)$ = -0.1 + 0.5 $V(s_0)$
$V(s_1)$ = -0.1
$V(s_2)$ = 1

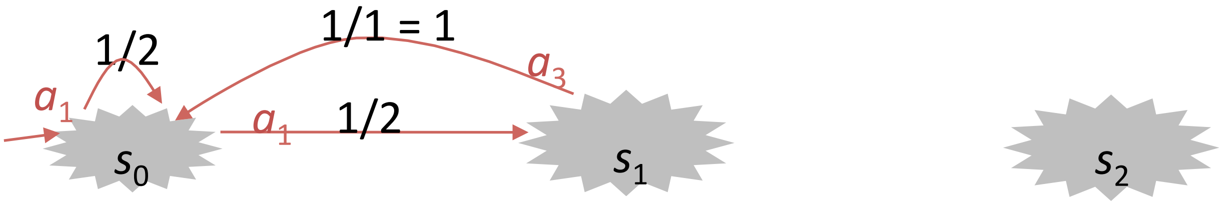
Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1}$

$V(s_0)$ = -0.1 + 0.5 (0.5 $V(s_0)$ + 0.5 $V(s_1)$)
$V(s_1)$ = -0.1
$V(s_2)$ = 1

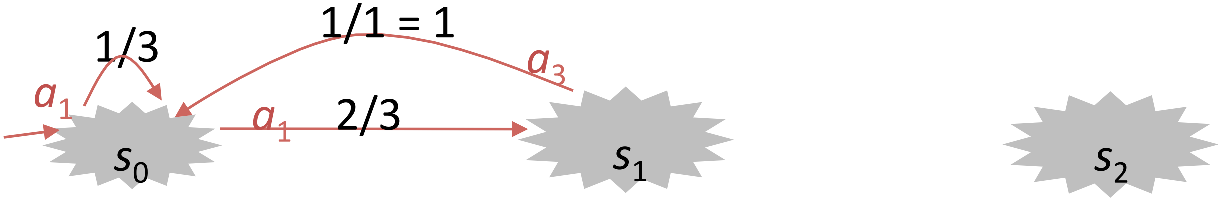
Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1}$

$V(s_0)$ = -0.1 + 0.5 (0.5 $V(s_0)$ + 0.5 $V(s_1)$)
$V(s_1)$ = -0.1 + 0.5 $V(s_0)$
$V(s_2)$ = 1

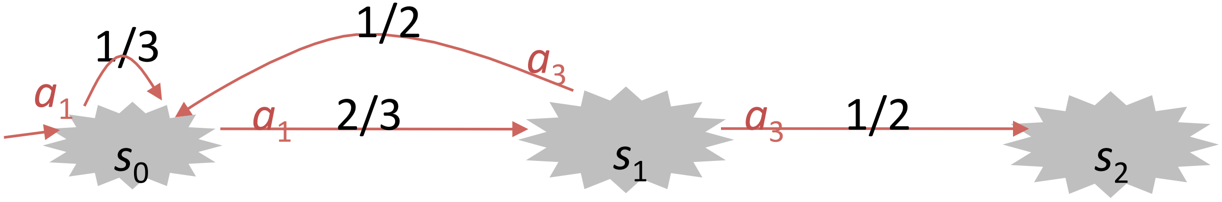
Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1}$

  $V(s_0) = -0.1 + 0.5\ (⅓\ V(s_0) + ⅔\ V(s_1)\ )$
  $V(s_1) = -0.1 + 0.5\ V(s_0)$
  $V(s_2) = 1$

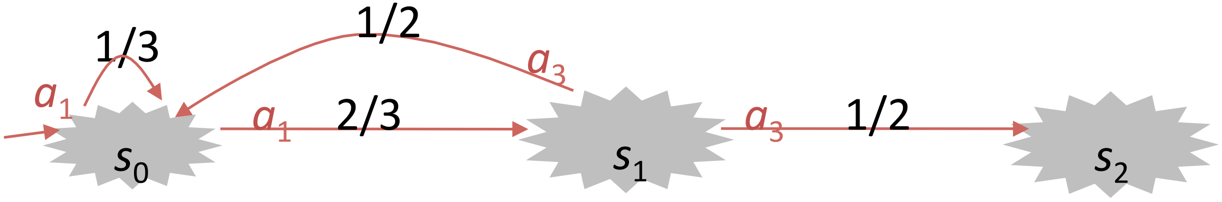
Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_2)_{1}$ fin de l'essai

  $V(s_0) = -0.1 + 0.5\ (⅓\ V(s_0) + ⅔\ V(s_1)\ )$
  $V(s_1) = -0.1 + 0.5\ (0.5\ V(s_0) + 0.5\ V(s_2)\ )$
  $V(s_2) = 1$

Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_2)_{1}$ $\boxed{\text{fin de l'essai}}$

$$\left.\begin{array}{l} V(s_0) = -0.1 + 0.5\ (⅓\ V(s_0) + ⅔\ V(s_1)\ ) \\ V(s_1) = -0.1 + 0.5\ (0.5\ V(s_0) + 0.5\ V(s_2)\ ) \\ V(s_2) = 1 \end{array}\right\}$$

à tout moment, on peut calculer les $V(s)$

Approche par programmation dynamique adaptative

- On a vu comment résoudre le système d'équations des $V(s)$
  - on peut écrire le système sous forme $b = A x$, et calculer $x = A^{-1} b$
- Cette opération peut être coûteuse à répéter après chaque observation
  - inverser la matrice A est dans $O(|S|^3)$

Approche par programmation dynamique adaptative

- Approche alternative: méthode itérative, similaire à *value iteration*
  1. Répéter (jusqu'à ce que le changement en V soit négligeable)
     I. pour chaque état s calculer:
        $$V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) \; V(s')$$
     II. si $|V - V'| \leq$ tolérance, quitter
     III. $V \leftarrow V'$

sans le max
p/r à l'action

- Plutôt qu'initialiser $V(s)$ à 0, on peut l'**initialiser à sa valeur précédente, avant la nouvelle observation**
  - une seule observation peut avoir un impact minime sur la nouvelle valeur de $V(s)$ qui en résulte
  - l'approche itérative + initialisation à la valeur précédente devrait donc converger rapidement

Approche par programmation dynamique adaptative

- Approche alternative: méthode itérative, similaire à *value iteration*
  1. Répéter (jusqu'à ce que le changement en V soit négligeable).
     I. pour chaque état s calculer:
        $$V'(s) = R(s) + \gamma \sum_{s' \in S} P(s' | s, \pi(s)) \; V(s')$$
     II. si $|V - V'| \leq$ tolérance, quitter
     III. $V \leftarrow V'$

- Autres accélérations
  - borner le nombre d'itérations (c.-à-d. ne pas attendre d'atteindre le seuil)
  - balayage hiérarchisé (*prioritized sweeping*)
    - » on garde un historique des changements $|V(s)- V'(s)|$
    - » on priorise les états s avec une grande valeur précédente de $|V(s)- V'(s)|$
- Permet de résoudre des problèmes où $|S|$ est beaucoup plus grand

Approche par programmation dynamique adaptative

- Contrairement à l'estimation directe, l'approche par PDA peut apprendre après chaque observation, c.-à-d. après chaque transition d'un essai
  - pas besoin de compléter un essai pour obtenir une nouvelle estimation de $V(s)$

- Parfois, la fonction de récompense n'est pas connue
  - l'agent ne fait qu'observer la récompense à chaque état, et n'a pas accès directement à la fonction $R(s)$
  - par contre on a besoin de $R(s)$ dans les équations de la fonction de valeur
  - dans ce cas, on initialise notre estimation $R(s)$ à 0, et
    on la met à jour lorsqu'on atteint l'état s
    pour la première fois

Approche par programmation dynamique adaptative

function PASSIVE-ADP-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, discount γ
    U, a table of utilities, initially empty
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    s, a, the previous state and action, initially null

  if s is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION(π, U, *mdp*)
  if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  return a