Chapitre 7: Optimisation par apprentissage

INF889B — Algorithmes d'optimisation combinatoire

Alexandre Blondin Massé

Université du Québec à Montréal

Hiver 2020

Partie b: Apprentissage par renforcement

Plan

Généralités

Éléments de base

- Apprentissage par **essais** et **erreurs**
- Similaire à ce que les **animaux** et les **humains** font
- L'agent **découvre** les actions plus intéressantes
- Mécanisme de **récompense** immédiates et futures
- L'agent doit équilibrer l'**exploitation** et l'**exploration**

Différence avec autres types d'apprentissage

- Pas **supervisé**: pas de données étiquetées, plus interactif, mais souvent combinés
- Pas **non supervisé**: ne cherche pas de structure cachée, mais maximise la récompense

Politique

- **Comportement** de l'agent à tout moment
- Fonction qui indique, pour chaque **état perçu**, quelle(s) **action(s)** devrait (pourraient) être effectuée(s):

$$\pi : \{\text{états possibles}\} \rightarrow \{\text{actions possibles}\}$$

- Peut être **déterministe** ou **stochastique**
- Peut être **précalculée** ou calculée en **temps réel**

Récompense

- En anglais, *reward signal*
- Rétroaction émise à **chaque étape**
- Généralement un **nombre réel**
- L'agent cherche à **maximiser** le total de ses récompenses
- **Pénalité**: souvent un nombre négatif
- Peut aussi être **nulle** (ni récompense ni pénalité)
- Mesure **locale**
- Utile pour **mettre à jour** la politique
- Peut être **déterministe** ou **stochastique**

Valuation d'un état

- En anglais, *value function*
- Fonction qui indique, pour chaque **état**, la **récompense totale** que l'agent peut s'attendre à obtenir:

$$v : \{\text{états possibles}\} \rightarrow \mathbb{R}$$

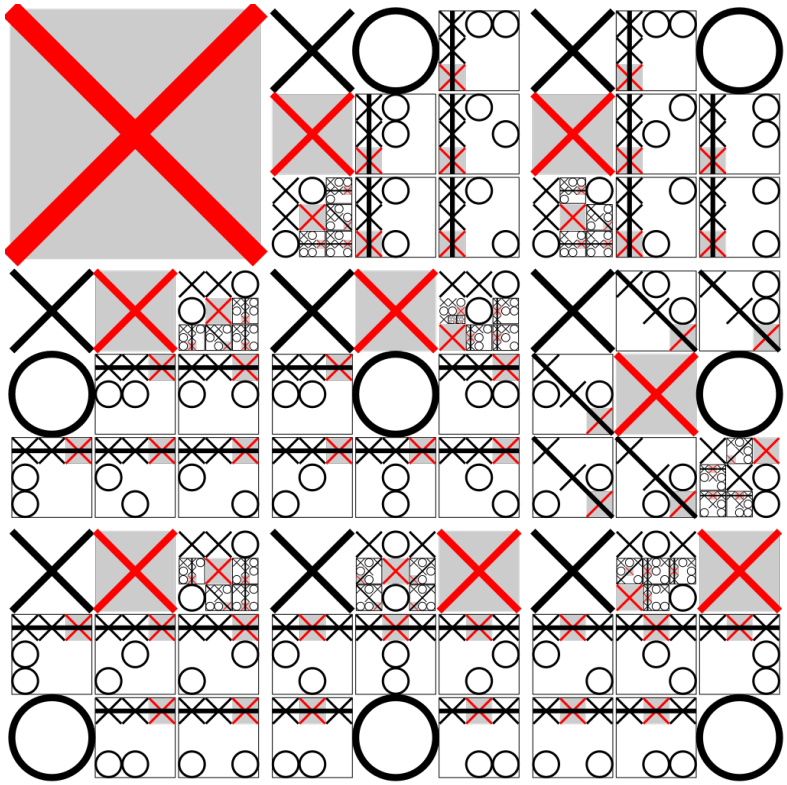
- Évaluation **globale**, complémentaire aux **récompenses**
- En général beaucoup plus difficile à évaluer qu'une **récompense**
- Doit souvent être **estimée**

Modèle

- Une **représentation** de l'environnement
- Peut être plus ou moins **riche**
- Est-il possible d'inférer des résultats sans faire les **actions**?
- **Avec modèle**: permet de planifier certains aspects
- **Sans modèle** (*model free*): on ne peut se fier qu'aux résultats des actions
- Possible de gérer ces deux cas
- **Exemple**: MuZero utilise un modèle encore plus général que AlphaZero

Exemple

Tic-tac-toe: stratégie parfaite (1/2)



(Source: Wikipedia)

- Si x commence dans un coin
- Décrite par Newell et Simon en **1972**:

Tic-tac-toe: stratégie parfaite (2/2)

1. **Gain**: si on peut gagner directement
2. **Bloc**: si l'adversaire menace de gagner directement
3. **Fourchette**: si on peut créer deux menaces de gagner simultanées
4. **Bloquer une fourchette**: s'il n'y en a qu'une possible, jouer là, s'il y en a plusieurs, jouer sur une case commune aux fourchettes, sinon, on joue un coup qui menace un gain immédiat (donc l'adversaire doit l'empêcher), de sorte qu'en défendant, l'adversaire ne crée pas de fourchette
5. **Centre**: jouer au centre, si la case est libre.
6. **Coin opposé**: jouer dans le coin opposé à l'adversaire, si la case est libre
7. **Coin**: s'il y a un coin de libre
8. **Côté**: s'il y a un côté de libre

Tic-tac-toe avec joueur imparfait

- Supposons qu'on joue contre un joueur imparfait
- Qui ne connaît pas la **stratégie parfaite**
- Par exemple, on peut le surprendre en jouant le premier coup dans un **coin** peut surprendre
- Mais qu'on se fixe comme objectif de **gagner**
- Autrement dit, une **partie nulle** est aussi mauvais que de **perdre**
- On obtient donc

$$\begin{cases} +1, & \text{si on gagne;} \\ -1, & \text{si on perd ou si on fait nulle.} \end{cases}$$

- Le comportement de l'adversaire n'est **pas connu**
- Comment pourrait-on **apprendre** à gagner?

Complexité du tic-tac-toe

- Soit $G = (V, A)$ le graphe orienté où V est l'ensemble des **états possibles** (à isométrie près) et $(s, s', j) \in A$ si et seulement s'il existe un coup du joueur j qui modifie l'état de s à s'
- On écrit $s \xrightarrow{j} s'$ ou $s \rightarrow s'$ (si j n'est pas important)
- Noter que G est un graphe acyclique orienté (en anglais, *DAG*)
- Voir Wikipedia pour plus d'informations

Questions

- Démontrez que $|V| = 765$
- Démontrez que le nombre de parties possibles est $26\ 830$. À quoi ce nombre correspond-il par rapport au graphe G?

Note

Vous pouvez démontrer les deux affirmations avec un programme informatique (plus facile) ou donner un argument mathématique (plusieurs cas à examiner).

Solution 1: minimax ou alpha-beta

Minimax

- Chaque joueur suppose que l'autre joueur joue **parfaitement**
- Pour chaque état s, soit $M(s) \in \{-1, +1\}$ le **meilleur score** qui peut être obtenu par le joueur qui joue le prochain coup
- Peut-être calculé **récursivement**:

$$M(s) = \max_{s \to s'} \min_{s' \to s''} M(s'')$$

Alpha-beta

- Pour chaque état, on maintient un intervalle $[\alpha(s), \beta(s)]$
- Qui donne des **bornes** sur les résultats possibles
- Ce qui nous permet d'**élaguer** l'espace de recherche
- Ne **change pas** le résultat, qui est le même que minimax

Solution 2: programmation dynamique

Supposons qu'on connaisse

- la **probabilité** que l'adversaire joue un coup donné
- à partir de **chaque état** possible

Alors on peut calculer

- par **programmation dynamique**
- le **coup** qui maximise les probabilités de gagner
- pour **chaque état**

Or, en pratique, on ne **connaît pas** ces probabilités

Plutôt, on peut **estimer** ces probabilités en jouant plusieurs parties avec l'adversaire

Solution 3: métaheuristiques

Recherche locale

- On considère une **politique** (déterministe) indiquant quel coup jouer pour chaque état
- **Qualité**: résultat d'une partie simulée quand on applique la politique
- **Voisinage**: on modifie un coup au hasard

Algorithme génétique

- **Individu**: une politique
- *Fitness*: résultat de la partie quand on applique la politique
- **Croisements** et **mutations**: différentes possibilités
- On cherche à **optimiser** la politique
- On pourrait intégrer de la **recherche locale**

Solution 4: apprentissage par renforcement

- On associe à chaque état une valeur $v(s) \in [0, 1]$
- estimant la probabilité de gagner à partir de s
- Initialement,

$$v(s) = \begin{cases} 1, & \text{si le joueur a complété une ligne;} \\ 0, & \text{si l'adversaire a complété une ligne;} \\ 1/2, & \text{sinon.} \end{cases}$$

- On simule une partie en jouant

$\rightarrow$ de façon **gloutonne** la plupart du temps (exploitation)

$\rightarrow$ au **hasard** en de rares occasions (exploration)

- À la fin de la simulation, pour chaque coup $s \rightarrow s'$ qu'on a choisi, on on met à jour la **valeur**:

$$v(s) \leftarrow v(s) + \alpha(v(s') - v(s))$$

où α est un paramètre qui tend (généralement) vers 0

Problème du bandit

Bandit à k bras

- On doit choisir parmi k **actions** différentes, désignées par a_1, a_2, ..., a_k
- Lorsqu'on choisit l'action a_i $(i = 1, 2, \ldots, k)$, on obtient une **récompense** R_t
- On **répète** ce choix t_{max} fois
- On ne **connaît pas** la distribution de probabilité de R_t
- Mais on sait qu'elle est **stationnaire**
- On cherche à maximiser le **total** des récompenses après t_{max} épisodes
- On dénote par A_t l'action **choisie** à l'étape t

Estimation

- Pour toute action a, sa **valeur** est définie par

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$$

- Idéalement, on devrait choisir a telle que $q_*(a)$ est **optimale**
- Mais on ne connaît pas q_*
- On va donc chercher **estimer** cette valeur
- Soit $Q_t(a)$ l'estimation qu'on va chercher à **apprendre**

Remarque

- R_t, A_t et Q_t sont des **variables aléatoires**
- Elles sont évidemment **dépendantes** l'une de l'autre
- Elles dépendent aussi de t

Exploitation et exploration

Stratégies à préciser

- Comment estimer Q_t?
- Comment choisir la prochaine action?

Plusieurs façons d'estimer Q_t

Dans tous les cas, on doit s'assurer

- d'**exploiter** l'information qu'on a obtenue
- d'**explorer** l'information qu'on ne connaît pas

Il faut trouver le bon **équilibre**

- c'est un **défi** en soi
- nous allons nous concentrer sur des versions assez **simples**

Calcul de la valeur d'une action

Estimation de Q_t

- On prend

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} \chi(A_i = a) R_i}{\sum_{i=1}^{t-1} \chi(A_i = a)}$$

où

$$\chi(p) = \begin{cases} 1, & \text{si } p \text{ est vrai;} \\ 0, & \text{sinon.} \end{cases}$$

- Autrement dit, on estime la moyenne Q_t par **échantillonnage**

Choix d'une action

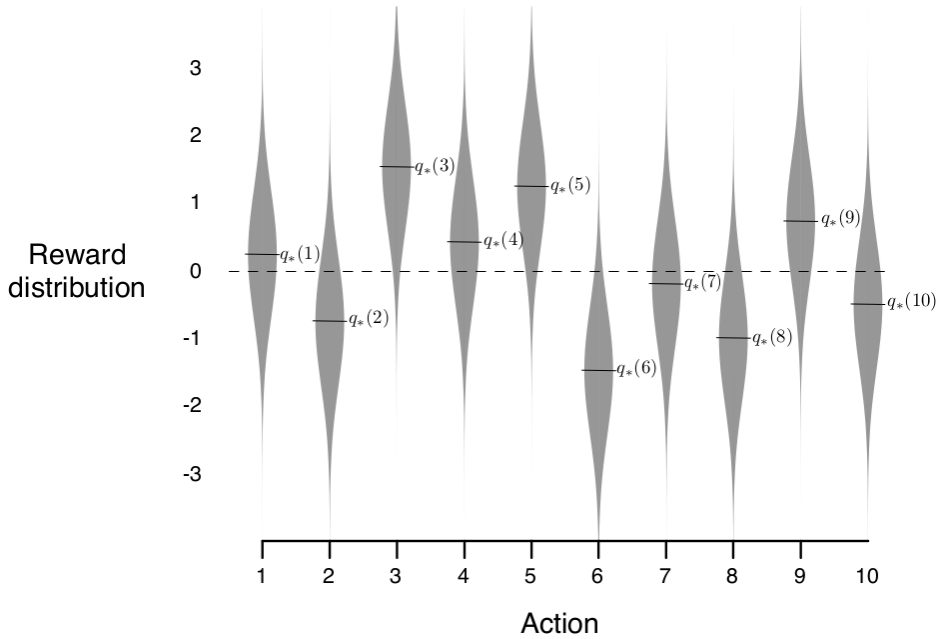
- Avec probabilité ε, on choisit une action au **hasard**
- Avec probabilité $1 - \varepsilon$, on choisit de façon **gloutonne**
- Stratégie appelée ε-gloutonne (en anglais, ε-*greedy*)

Pseudocode

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + (r - q(a))/n(a)$

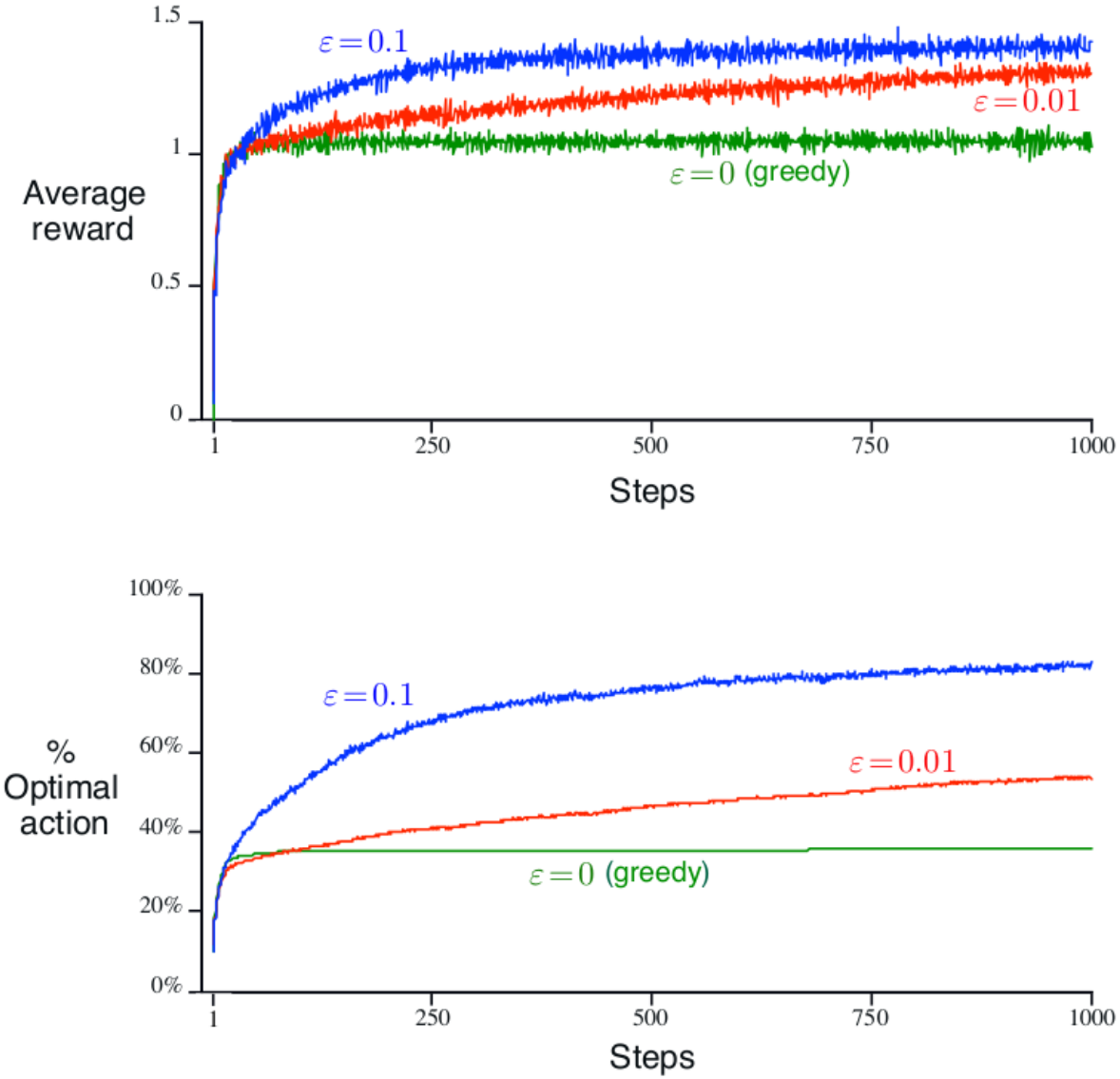
Jeu de données pour tests

- **Sutton et Barto** ont généré aléatoirement $2\,000$ exemplaires
- En fixant $k = 10$ actions possibles
- En choisissant, pour $i = 1, 2, \ldots, k$, le paramètre μ_k selon $\mathcal{N}(0, 1)$
- Puis en posant $R_t \sim \mathcal{N}(\mu_k, 1)$



(Source: Sutton et Barto, chapitre 2)

Simulation



(Source: Sutton et Barto, chapitre 2)

Mise à jour de la valeur

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + (r - q(a))/n(a)$

- On peut généraliser la **mise à jour**

Mise à jour de la valeur

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + \alpha_n(r - q(a))$

- On peut généraliser la **mise à jour**
- En autant que α_n respecte certaines **contraintes**

Convergence

Théorème

L'algorithme converge avec probabilité 1 si

$$\sum_{n \geq 1} \alpha_n(a) = \infty \quad \text{et} \quad \sum_{n \geq 1} \alpha_n^2(a) < \infty$$

Cas non stationnaire

- Préférable **« d'oublier »** partiellement les récompenses passées
- On prend α **constant**
- Mais ne **converge pas**

Choix d'action optimiste dans l'incertitude

Stratégies gloutonnes

- On a vu qu'on peut choisir de façon **gloutonne**
- Ou de façon ε-**gloutonne**

Optimiste dans l'incertitude

- En anglais, *UCB* = *Upper-Confidence-Bound*
- Tient compte des **meilleures** estimations jusqu'à maintenant
- Mais aussi de l'**incertitude** (quand $n(a)$ est petit)
- Se traduit par l'affectation

$$a \leftarrow \text{argmax}_a \left[q(a) + c\sqrt{\frac{\ln t}{n(a)}} \right]$$

  où c est un **paramètre** à choisir
- Voir **pseudocode** modifié à la diapositive suivante

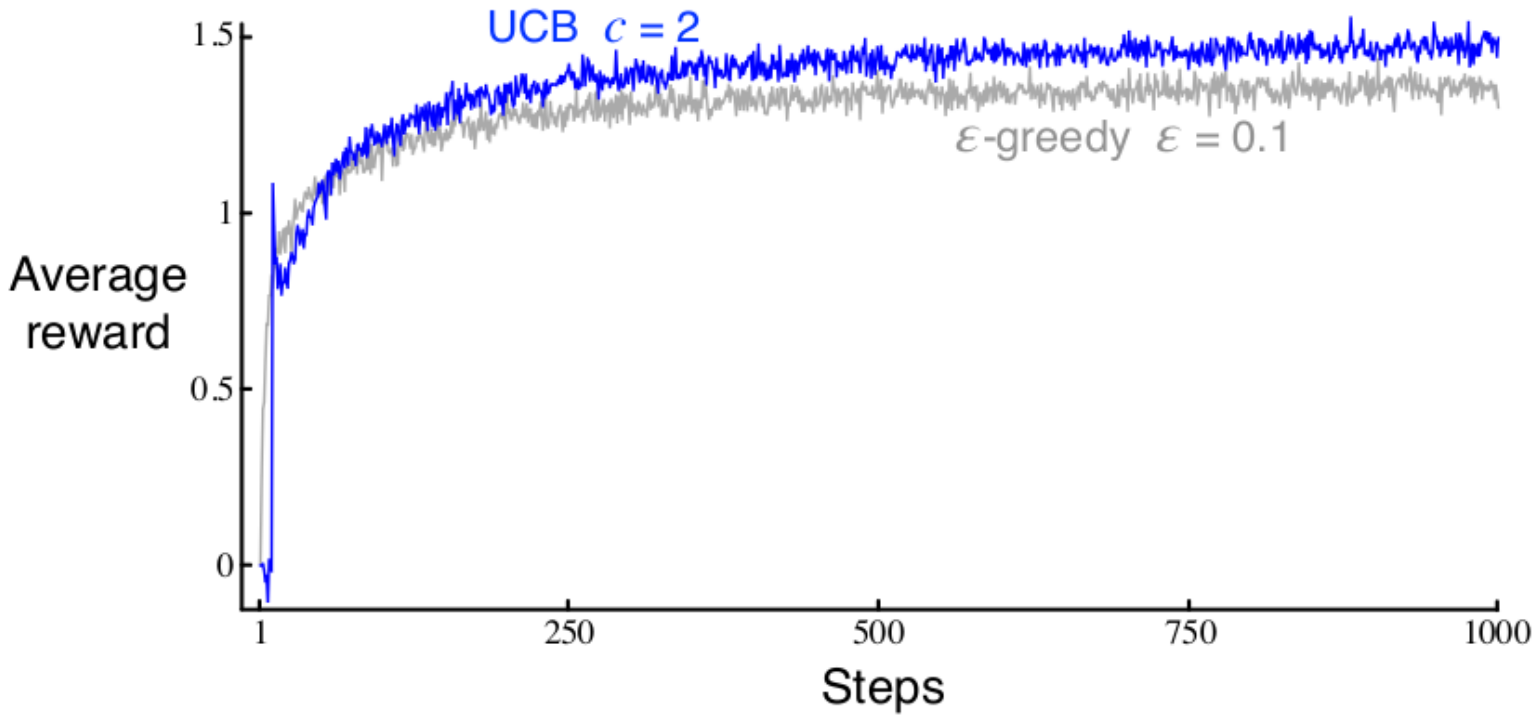
Optimiste dans l'incertitude

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + \alpha_n(r - q(a))$

Optimiste dans l'incertitude

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $a \leftarrow \text{argmax}_a \left[q(a) + c\sqrt{\frac{\ln t}{n(a)}} \right]$
5: Soit r la récompense retournée après avoir choisi a
6: $n(a) \leftarrow n(a) + 1$
7: $q(a) \leftarrow q(a) + \alpha_n(r - q(a))$

Simulation



(Source: Sutton et Barto, chapitre 2)

- Efficace quand la distribution est **stationnaire**