Exploration en ligne

IFT-17587

Concepts avancés pour systèmes intelligents

Luc Lamontagne

Plan de la présentation

- **Exploration en ligne**
- **LRTA***

Retour sur l'exploration

- Les algorithmes étudiés dans le cours sont de type hors ligne (*offline search*)
  - Par ex. en largeur, en profondeur, A*, SMA*…
- Résolution de problèmes **hors ligne**
  - L'agent a en mémoire une représentation complète du problème à résoudre.
  - Il trouve une solution complète avant d'exécuter les actions.
  - Il n'utilise pas les nouvelles perceptions.
- Adéquat pour tous les environnements ?

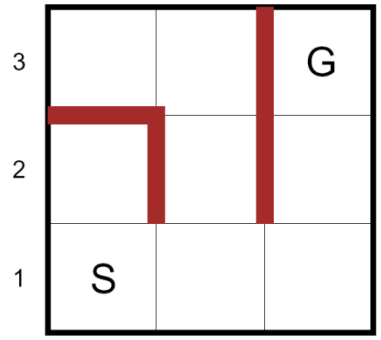
Exploration en ligne (*online search*)

- En ligne → On alterne les calculs et les actions
  - On commence par exécuter une action.
  - On observe l'environnement.
  - On fait le choix d'une prochaine action.
  - Et on répète jusqu'à ce qu'on atteigne un but.
- Une bonne idée pour certains types d'environnements
  - Dynamique
    - Contrainte de temps – pénalité si trop de temps s'écoule.
  - Stochastique ou non déterministe
    - Prévoir toutes les éventualités possibles – intractable
    - Ex. planifier complètement une partie d'échec avant d'avoir jouer le premier coup.
  - Partiellement observable
    - L'agent ne connaît pas tous les états et actions possibles.
    - Par ex. un robot qui se déplace dans un immeuble qu'il n'a jamais visité auparavant.

Exploration en ligne :
Formulation du problème

- L'agent doit exécuter des actions pour résoudre le problème.

- Il ne connaît que les éléments suivants :
  - Actions(*s*) : les actions permises dans l'état *s*
  - c(*s*,*a*,*s'*) : le coût d'une action
    - N'est connu que lorsque l'agent atteint l'état *s'*
  - But(*s*) : détermine si un état est un but.



- Dans le pire cas, l'agent ne peut connaître les successeurs que par l'exécution d'actions
  - Mais ce degré d'ignorance peut être réduit pour certaines applications.
    - Localisation des obstacles seulement.
  - Mais l'agent peut toujours reconnaître un état lorsqu'il l'occupe.
  - Et il a une heuristique h(*s*) pour estimer la distance du but.

Exploration en ligne :

Agent d'exploration en ligne

- Après chaque action, l'agent reçoit un percept *p*.
  - Le percept indique l'état qui *s'* est atteint.
  - L'agent augmente sa description de la carte de l'environnement (*result[a,s]*).
    - On garde en mémoire le résultat d'appliquer l'action *a* pour *s*.
  - A partir de cette carte, il décide où aller.
    - Il doit développer le nœud *s'* qu'il occupe.
      - Pas possible de choisir le nœud de moindre coût comme A*.
    - Il choisi une action *a* qui n'a pas encore été explorée (*unexplored*).
    - Si l'agent a essayé toutes les actions pour un état *s'*
      - Il retourne à l'état prédécesseur *s.*
      - Les états prédécesseurs sont conservés dans une liste (*unbacktracked*).
- Similaire à faire une recherche en profondeur (*DFS*).

Exploration en ligne :

Agent d'exploration en profondeur

```
function ONLINE-DFS-AGENT(s') returns an action
  inputs: s', a percept that identifies the current state
  static: result, a table, indexed by action and state, initially empty
          unexplored, a table that lists, for each visited state, the actions not yet tried
          unbacktracked, a table that lists, for each visited state, the backtracks not yet tried
          s, a, the previous state and action, initially null

  if GOAL-TEST(s') then return stop
  if s' is a new state then unexplored[s'] ← ACTIONS(s')
  if s is not null then do
      result[a, s] ← s'
      add s to the front of unbacktracked[s']
  if unexplored[s'] is empty then
      if unbacktracked[s'] is empty then return stop
      else a ← an action b such that result[b, s'] = POP(unbacktracked[s'])
  else a ← POP(unexplored[s'])
  s ← s'
  return a
```

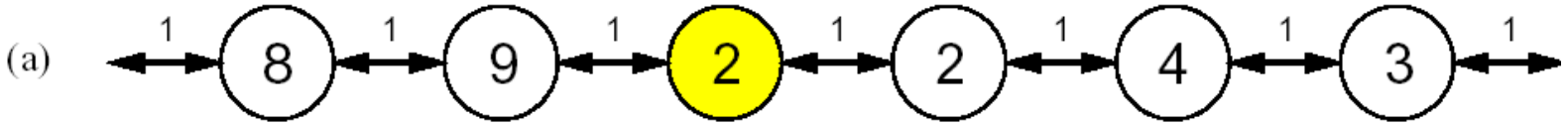
Exploration en ligne :

Algorithmes d'exploration en ligne

- L'algorithme d'escalade (*hill climbing*) est un algorithme en ligne.
  - Malheureusement pas très utile dans sa version de base.
    - On peut rester pris dans un minimum local.
    - Par de redémarrage aléatoire possible non plus.
- Pour se sortir d'un minimum local :
  - On peut utiliser une marche aléatoire (*random walk*) .
    - Garantie théorique de trouver un but.
    - Mais il faut être patient.
  - Il est préférable d'ajouter une mémoire pour s'en sortir.
    - On met à jour le coût $H(s)$ selon les expériences de l'agent.
    - Si on se retrouve dans un minimum, on rebrousse chemin dans la meilleure direction disponible.
    - Donc passer par s' qui minimise $c(s,a,s')+ H(s')$
- Une version de ce type d'algorithme → LRTA*
  - *Learning real-time A*.*
  - 2 composantes: règles de sélection de l'action et de mise à jour.
  - Peut explorer un espace de n états en $O(n^2)$ en pire cas.

Exploration en ligne :

Algorithmes d'exploration en ligne



Exploration en ligne :

*Learning Real-Time A** (LRTA*)

```
function LRTA*-AGENT(s') returns an action
  inputs: s', a percept that identifies the current state
  static: result, a table, indexed by action and state, initially empty
          H, a table of cost estimates indexed by state, initially empty
          s, a, the previous state and action, initially null

  if GOAL-TEST(s') then return stop
  if s' is a new state (not in H) then H[s'] ← h(s')
  unless s is null
      result[a, s] ← s'
      H[s] ← min_{b ∈ ACTIONS(s)} LRTA*-COST(s, b, result[b, s], H)
  a ← an action b in ACTIONS(s') that minimizes LRTA*-COST(s', b, result[b, s'], H)
  s ← s'
  return a

function LRTA*-COST(s, a, s', H) returns a cost estimate
  if s' is undefined then return h(s)
  else return c(s, a, s') + H[s']
```