Agents intelligents

Déf : entité qui peut percevoir son environnement à l'aide de ses capteurs; et agir dans son environnement à l'aide de ses effecteurs.

Son intelligence peut être vu comme une fonction qui associe un historique à une action. Intelligence d'un agent = Architecture + Programme.

Ex : Rover de la NASA, Radarsat-II de l'ASC, Mario de nintendo.

Agents rationnels : doit agir « correctement » en fonction de ce qu'il perçoit et de ses capacités d'action, il devrait choisir une action qui maximise la mesure de performance.

Un agent est autonome si son comportement est déterminé sans interventions externes et par sa propre expérience.

Modèle PEAS

(PEAS = Performance, Environnement, Actuateurs, Senseurs)

-Mesure de **p**erformance;

-Connaissance de l'**e**nvironnement;

-Les actions que l'agent peut effectuer (**a**ctuateurs) ;

-La séquence des perceptions par les **s**enseurs de l'agent.

Types d'environnement

-Observable (vs.partiellement observable): Grâce à ses senseurs, l'agent a accès un état complet de l'environnement à chaque instant.

-Déterministe (vs.stochastique): suivant de l'environnement est entièrement déterminé par l'état courant et l'action effectuée par l'agent.

-Épisodique (vs.séquentiel): Les opérations de l'agent sont divisés en épisodes: chaque épisode consistant à observer et effectuer une seule action ; et le choix de chaque action dans un épisode ne dépendant que de cet épisode.

-Statique (vs.dynamique): l'environnement ne change pas lorsque l'agent n'agit pas.

-Discret (vs.continu): Un nombre limité et clairement distincts de données sensoriels et d'actions.

-Agent unique (vs.multi-agent): Un agent opérant seul dans un environnement.

Un agent est complètement spécifié par sa fonction agent qui associe une séquence de données sensorielles à une séquence d'actions.

Types d'agent

Quatre types ordonnés en ordre de généralité: -Simple reflex

-Model-based reflex

-Goal-based

-Utility-based

Résolution des problèmes par recherche

Formalisation d'un problème de recherche :

-Entrées : -Un état initial n0.

-Une fonction but(n) qui retourne TRUE ssi le but est atteint dans le n.

-Une fonction successeurs(n) qui génère les successeurs de n.

-Sortie : -Solution (séquence/ actions)

Le coût solution est la somme des coûts des arêtes dans le graphe.

(Il peut y avoir plusieurs états satisfaisant le but.)

Critères d'évaluation des algorithmes de recherche

-Correcte : La solution correcte.

-Complétude : trouve une solution s'elle existe et indique l'absence des solutions lorsqu'elle n'existe pas

-Optimalité : solution retournée est optimale ou proche optimale.

-Complexité temporelle.

-Complexité spatiale.

Algorithmes de recherche

Recherche non informée :

Recherche en largeur :

-Utilise les files (FIFO) ;

-Dans une file OPEN : enfiler le nœud initial n0

-Enfiler dans OPEN les fils de n0

-Successeurs(nœud) : Visiter chaque fils et enfiler leurs fils dans open jusqu'à atteindre le but.

Recherche en profondeur :

-Utilise les piles (LIFO) ;

Pour les arbres :

-Dans une pile OPEN : empiler le nœud initial n0

-Dans OPEN: empiler les fils de n0

- Successeurs(nœud) : dépiler OPEN

-Réappliquer l'algorithme sur chaque nœud dans OPEN

Pour les graphes :

-Dans une pile OPEN : empiler le nœud initial n0

- Dans CLOSED (ou VISITED): stocker n0

-Dans OPEN: empiler les fils de n0

- Successeurs(nœud) : dépiler OPEN

-- Dans CLOSED (ou VISITED): stocker le nœud dépilé.

-Réappliquer l'algorithme sur chaque nœud dans OPEN et non visité

La différence entre les arbres et les graphes est que dans un arbre, il n'y a pas de cycles, et chaque nœud a un seul parent, mais sur un graphe l'algorithme doit maintenir une liste de nœuds visités pour éviter de boucler indéfiniment dans un cycle.

Recherche en profondeur avec coûts uniformes :

-Même algorithme en profondeur mais on donne importance aux couts.

Recherche en profondeur avec profondeur limitée :

-Même algorithme en profondeur mais on précise un certain niveau de profondeur qu'on ne dépasse pas.

Recherche en profondeur itérative:

-Même algorithme en profondeur mais il garantit que tous les nœuds à une profondeur donnée sont explorés avant de passer à la profondeur suivante.

Recherche informée :

Algorithme meilleur en premier (best-first-search):

-Choisit le nœud prochain qui 'semble' le plus proche du but, fait par une fonction heuristique.

- **Une heuristique :** méthode qui calcule rapidement une solution approximative/incomplète à un problème généralement complexe.
- Estimation de la distance entre un nœud n et un but G.
- Notée h(n) : estime le cout restant pour atteindre le but implicite G depuis un nœud n.
- **Exemples pour la navigation sur une carte:** - distance euclidienne (vol d'oiseau)
    - Distance de Manhattan

Le best first search:

-Dans OPEN on stocke n0 (état initial).

-Sélection du nœud : Choisir le nœud dans OPEN qui a la meilleure valeur de fonction d'évaluation. La fonction d'évaluation est souvent notée f(n)=g(n)+h(n), où g(n) est le coût cumulé jusqu'au nœud n et h(n) est la valeur heuristique.

-Retourner la solution si le but est atteint, sinon ajouter les successeurs de n dans OPEN et répéter l'opération.

-Fin : l'algorithme se termine lorsqu'une solution est trouvée ou lorsque tous les nœuds ont été atteint.

Algorithme greedy-best-first-search:

-Dans OPEN on stocke n0 (etat initial).

-h(n): Choisit le nœud avec la meilleure valeur heuristique comme le prochain nœud à explorer.

-Retourner la solution si le but est atteint, sinon ajouter les successeurs de n dans OPEN et répéter l'opération.

-Fin : l'algorithme se termine lorsqu'une solution est trouvée ou lorsque tous les nœuds ont été atteint.

La différence entre le BFS et le GBFS réside dans la façon dont ils sélectionnent le prochain nœud à explorer.

Algorithme A* (A-etoile):

Déf: Extension de Djikstra utilisant une fonction heuristique.

Clés importants :

-OPEN : contient les nœuds à traiter, ordonnés par une fonction f(n).

-CLOSED : contient les nœuds déjà traités/ visités.

-F(n) : pour chaque n, elle estime le coût un chemin de n0, passant par n, et arrivant à g satisfaisant le but.
f(n)=g(n)+h(n), avec : -g(n) : coût réel du chemin optimal, partant de n0 jusqu'au nœud n courant.

-h(n) : fonction heuristique estime le coût du reste du chemin de n jusqu'à un nœud g satisfaisant le but.

Algorithme :

-Créer une liste OPEN ordonnée par f(n), et une liste CLOSED.

-Dans OPEN : insérer n0.

-Sélection : le premier nœud de OPEN.

-Si le but est atteint, l'algorithme se termine, sinon on génère les successeurs du nœud sélectionné et on les insère dans OPEN (uniquement s'ils ne sont pas déjà présents dans OPEN ou CLOSED) avec une mise à jour des valeurs de g(n) et h(n).

-Un fois fini avec le nœud courant, on le stocke dans CLOSED.

-Répéter les étapes depuis la sélection jusqu'à ce qu'une solution est trouvée ou que tous les nœuds sont explorés.

Propriétés de A* :

-Si le problème n'a pas de solution, tous les nœuds seront explorés. Une fois OPEN est vide il constate directement qu'aucune solution existe.

-Si le problème a une solution A* la trouve toujours.

-A* utilisé avec une heuristique admissible, retourne toujours une solution optimale (s'elle existe).

Recherche locale

-La solution est seulement un état.

-Trouve une configuration satisfaisant le but.

-Utilise une fonction objective (qui ne considère pas le chemin), exclusivement basée sur les caractéristiques de l'état.

- Garde seulement certains états visités en mémoire.

- Généralement, une fonction objective doit être optimisée (maximisée ou minimisée).

-Ne garantit pas de solution optimale. Sa capacité est de trouver une solution acceptable rapidement.

La méthode de l'escalade (hill-climbing):

Entrée : -état initial.

-Fonction à optimiser : notée VALUE, parfois notée h(n).

Méthode :

-Le nœud courant est initialisé à l'état initial.

-Itérativement le nœud courant est comparé à ses successeurs immédiats. Le meilleur successeur immédiat (celui qui a la plus grande valeur VALUE que le nœud courant), devient le nœud courant.

-Si un voisin n'existe pas on retourne le nœud courant comme solution.

Variantes :

-Stochastique : choisit au hasard l'état suivant parmis les successeurs

-Du premier choix : utile quand il y a un grand nombre de successeurs. Génère les

successeurs immédiats aléatoirement et s'escalade dès que un d'eux est meilleur que l'état courant.

-Avec reprise aléatoire : Quand on a trouvé un optimum local, on relance la recherche avec un nouvel état tiré aléatoirement. Probablement complet.

La méthode du recuit simulé (simulated annealing):

-Amélioration de l'escalade réduisant le risque d'être piéger dans les optimums locaux.

-Permet à l'algorithme d'accepter occasionnellement des solutions pires que la solution actuelle dans l'espoir de trouver un minimum global plutôt que de rester coincé dans un minimum local.

Méthode :

-On choisit le meilleur voisin ou un voisin moins bon immédiat.

-Sélectionner un voisin.

-Calculez la différence de coût entre la nouvelle solution et la solution actuelle.

- Si la nouvelle solution est meilleure ($\Delta E<0$), acceptez-la comme la nouvelle solution courante. Sinon, acceptez-la avec une certaine probabilité qui diminue au fil du temps selon une fonction d'acceptation probabiliste basée sur ΔE et la température T.

-Répéter jusqu'à atteindre une contrainte.

- Diminuez la température (T) conformément à une fonction de refroidissement prédéfinie (aide à accepter moins fréquemment des solutions pires).

-Fin : lorsqu'un critère est atteint.

Recherche Tabou (Tabu Search)

-Enregistre n derniers les états visités dans l'ensemble tabou.

-n est choisi expérimentalement/selon la mémoire disponible, ce qui réduit les oscillations.

Recherche en faisceau (beam searh)

- On fait progresser un ensemble de n états au lieu d'un seul.

-Début : un ensemble de n nœuds choisi aléatoirement.

-A chaque itération tous les successeurs de n sont générés

-Si un de ces successeurs satisfait le but on arrête. Sinon on prend les n meilleurs de ces états et on recommence.

Algorithmes génétiques

- On commence avec une population de n états aléatoires : Première génération.

-Chaque état est présenté avec une chaine de symbole fini (par exemple {0,1})

-A une fonction d'évaluation qu'on cherche à maximiser (fonction fitness).

-Pour créer de nouvelles générations on applique la sélection, le croisement et mutation.

- Sélection : en donnant une probabilité plus élevée de sélection aux individus de meilleure qualité (avec une meilleure valeur de fitness).

-Croisement : Le croisement simule la reproduction en combinant des parties de deux parents pour créer des descendants.

-Mutation : La mutation introduit des changements aléatoires dans les gènes d'un individu pour explorer de nouvelles régions de l'espace de recherche.

Spécification des problèmes

(exercices d’examen)

Pour spécifier un problème, on indique les composants suivants :

-État Initial (Si) : C'est la configuration de départ du problème.

-État Final (Sf) : C'est la configuration que vous souhaitez atteindre.

-Opérateurs : les actions autorisées pour passer d'un état à un autre.

-Contraintes : définissent les règles ou restrictions qui s'appliquent aux opérations.

-Objectif : est l'état final que vous souhaitez atteindre en appliquant les opérateurs autorisés tout en respectant les contraintes.