Résolution de problèmes en intelligence artificielle et optimisation combinatoire : les algorithmes A*

Michel Couprie

Le 5 avril 2013

Ce document est une courte introduction à la technique dite A*. Pour aller plus loin, on se reportera à [Nilsson].

1 Exemple 1 : le problème des 8 reines

On se place dans le cas d'un échiquier standard (de taille 8×8). Le problème est de placer 8 reines sur cet échiquier de telle sorte qu'aucune reine ne peut se faire prendre par une autre (une reine peut capturer toutes les pièces se trouvant sur sa rangée, sa colonne ou ses diagonales).

La figure 1 page suivante présente le placement de 2 reines sur un échiquier. On voit que le nombre de cases disponibles pour le placement de 6 autres reines est très restreint.

Voici deux approches possibles pour le placement des huit reines :

1. Recherche aveugle : On place la n-ième reine sur une case choisie aléatoirement parmis les cases libres (une case est dite libre si elle n'est ni occupée, ni "menacée" par une reine). A un moment, il n'y aura plus de cases libres pour le placement des reines. S'il reste des reines à placer, il y aura alors remise en cause d'un ou de plusieurs choix faits précédemment (retour arrière).
2. Recherche heuristique : On place la n-ième reine sur une des cases restées libres, de telle manière que le nombre de cases "menacées" (susceptibles d'être prises par cette reine) soit minimal. Autrement dit, on cherche à maximiser à chaque étape le nombre de cases libres restantes.

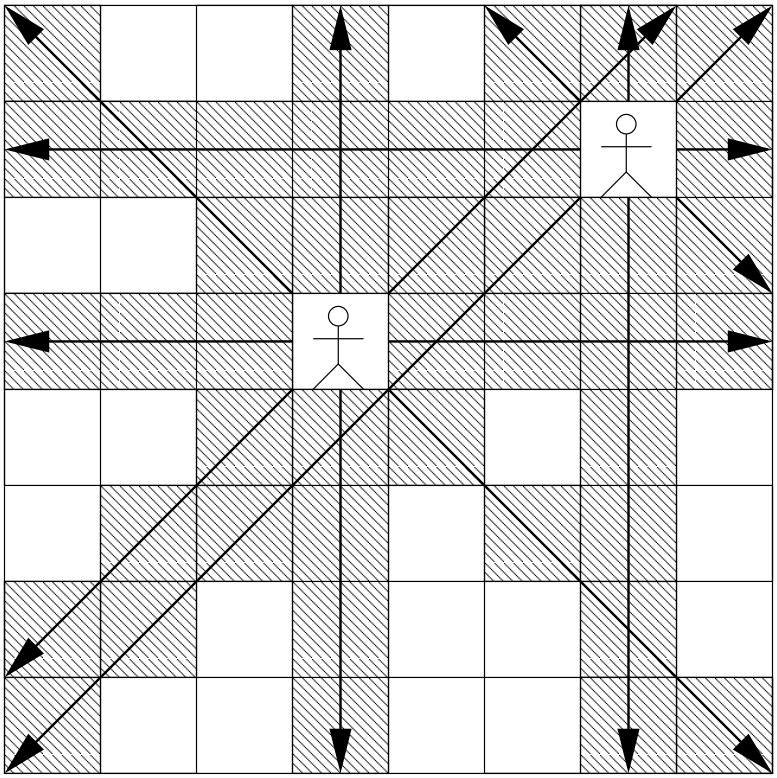

FIG. 1 – Exemple de placement pour le jeu des 8 reines

2 Graphe de résolution de problème

Les problèmes que l'on cherche à résoudre par cette approche consistent à faire évoluer un système, d'un état dit initial à l'un des états dits terminaux ou buts, au moyen d'un ensemble de règles.

Dans l'exemple précédent, le système est un échiquier portant un certain nombre de reines. L'état initial est l'échiquier vide, les états terminaux sont les configurations de 8 reines ne se menaçant pas mutuellement.

Un **graphe de résolution de problème** ou **GRP** est un graphe dont les sommets sont les états possibles du problème. Il y a un arc $u = (i, j)$ dans le graphe si une règle permet de passer de l'état i à l'état j. On associe un coût $c(u)$ à cet arc. On doit distinguer le sommet initial et les sommets terminaux.

Formellement, un graphe de résolution de problème est un quintuplet $(S, \vec{A}, d, B, c)$:
– S est l'ensemble des sommets du graphe (ou états du problème),
– $\vec{A}$ l'ensemble des arcs,
– $d \in S$ le sommet de départ (état inital du problème),
– $B \subset S$ l'ensemble des sommets buts,
– $c : \vec{A} \longrightarrow \mathbb{R}$, où $c(i, j)$ est le coût de l'application de la règle permettant de passer de

l'état i à l'état j.
Pour un même problème, on peut en général associer plusieurs GRP.

Dans l'exemple des 8 reines, on peut prendre pour états toutes les combinaisons de $0, 1, 2, \ldots, 8$ reines telles que les reines ne se menacent pas mutuellement. Le nombre de telles configurations est de l'ordre du million.

Toutefois, on peut voir que l'on ne réduit pas la généralité du modèle en contraignant la première reine placée à se trouver sur la première rangée, la seconde sur la seconde rangée, etc, puisqu'une solution doit obligatoirement comporter une reine sur chaque rangée.

Les états sont nettement moins nombreux avec ce second modèle, leur nombre est de l'ordre de quelques milliers.

Un graphe de résolution de problème est souvent énorme, parfois même infini, ainsi le but de la recherche heuristique est d'explorer partiellement le graphe pour aboutir à une solution (c'est à dire trouver un plus court chemin de i à un sommet $s \in B$), en cherchant à limiter le plus possible la partie explorée.

3 Exemple 2 : jeu du taquin

Le jeu du *taquin* est un jeu de plateau dans lequel il faut passer d'une configuration de départ à une configuration d'arrivée sachant que les cases numérotées ne peuvent se déplacer que sur une case vide immédiatement adjacente.

	1	3
7	5	8
4	2	6

Configuration de départ

1	2	3
4		5
6	7	8

Configuration d'arrivée

FIG. 2 – Jeu de taquin

Les contraintes étant :
– Passer de la configuration de départ à la configuration d'arrivée
– Minimiser le nombre de déplacements
Un graphe de résolution de problème possible associé à ce jeu est celui tel que :
– les sommets du GRP sont les états possibles du jeu
– le sommet initial est la configuration de départ
– le sommet but (unique) est la configuration d'arrivée
– il existe un arc de la configuration x à la configuration y si un mouvement autorisé permet de passer de x à y.

Remarque : Ce GRP possède à la fois des cycles : il peut exister 2 chemins différents pour passer d'un sommet x à un sommet y ; et des circuits : les règles permettent aussi de passer d'une configuration x à y puis de revenir à x.

4 Stratégies de recherche

4.1 Stratégie sans mémoire : la recherche arrière (*backtrack*)

Cette stratégie consiste à explorer le GRP via un chemin issu de d jusqu'à atteindre
– le but (c'est gagné)
– un puits (c'est perdu)
– une profondeur limite fixée d'avance
Si le but n'est pas atteint, on revient au dernier choix effectué chronologiquement (en *oubliant* la partie venant d'être explorée). Pour eviter les circuits on mémorise les états du graphe se trouvant sur le chemin courant. Tout nouvel état atteint est comparé à la liste des états du chemin.

Cette stratégie pose toutefois des problèmes : si le but ne peut être atteint dans la limite de profondeur fixée, il faut augmenter celle-ci et tout recommencer.

4.2 Stratégie avec mémoire : la recherche avec graphe (RAG)

On a un GRP qui est défini implicitement, en général trop grand pour être representé en mémoire et être exploré en totalité.

On va développer (c'est à dire représenter explicitement) une partie du GRP suffisante pour trouver le sommet but. On évitera ainsi les explorations redondantes.

4.2.1 Méthode (d'après [Nilsson])

1. **Créer un graphe de recherche** G qui consiste uniquement en un sommet de départ d. Mettre d sur une liste appelée OUVERT.
2. **Créer une liste appelée** FERME qui est initialement vide.
3. **BOUCLE**, si OUVERT est non-vide, sinon échec.
4. **Sélectionner le premier sommet d'OUVERT**, l'enlever d'OUVERT, et le mettre dans FERME, Appeler ce sommet n.
5. **Si** n **est un sommet but, terminer la procédure** avec succès.
6. **Développer le sommet** n, produisant l'ensemble M de ses successeurs et les mémoriser comme successeurs de n dans G.
7. **Mémoriser un pointeur vers** n **à partir des éléments de** M qui n'étaient pas déjà dans G (c'est à dire pas déjà dans OUVERT ou FERME). Ajouter ces éléments de M à OUVERT. Pour chaque élément de M qui était déjà dans OUVERT ou FERME, décider si l'on redirige ou non le pointeur vers n (voir ci-dessous). Pour chaque membre de M déjà dans FERME, décider pour chacun de ses descendants dans G si l'on redirige ou non le pointeur.
8. **Réordonner la liste OUVERT**, soit arbitrairement, soit selon des heuristiques.
9. **Aller à BOUCLE**

Les *pointeurs* dont il est question à l'étape 7 servent à retrouver, lorsqu'un sommet but b est atteint, un plus court chemin de d à b. Il suffit de "remonter" à partir de b en suivant les pointeurs.

Les heuristiques utilisées à l'étape 8 peuvent s'appuyer sur des connaissances spécifiques au problème.

4.3 Algorithmes A et A^*

Dans la recherche avec graphe (RAG), le choix de la fonction d'évaluation f pour le tri de OUVERT s'avère être un point crucial. Dans le cas de la méthode appelée **algorithme A**, on choisit f telle que pour tout sommet n du GRP, $f(n)$ estime le coût d'un chemin optimal de d à un but passant par n.

L'application f peut se mettre sous la forme :

$$f(n) = g(n) + h(n)$$

– $g(n)$ estime le coût d'un chemin optimal de d à n.
– $h(n)$ estime le coût d'un chemin optimal de n à un but.
Posons $g^*(n)$ le coût d'un chemin optimal dans le GRP de d à n et $h^*(n)$ le coût d'un chemin optimal de n à un but. On a $f^*(n) = g^*(n) + h^*(n)$. On veut que f estime (au mieux) f^*.

Pour $g(n)$ il est naturel de prendre le coût de d à n dans le graphe de recherche (partie du GRP déjà developpé). Pour $h(n)$ on s'appuie sur l'information spécifique au problème. On nomme h fonction heuristique.

Par exemple dans le cas du jeu de Taquin, on pourra prendre :
– $g(n)$ = distance (nombre d'arcs) de d à n dans le graphe de recherche.
– $h(n)$ = nombre de cases mal placées par rapport à la configuration but.
Si pour tout sommet n, $h(n) \leq h^*(n)$, alors on démontre que l'algorithme **A** trouvera un chemin optimal de d à un but (s'il en existe). Dans ce cas, on parle d'**algorithme A***. Dans le cas particulier où $h = 0$, on a bien un algorithme **A*** qui n'est autre, si les coûts des arcs du GRP sont unitaires, que l'exploration en largeur.

5 Bibliographie

[Cormen] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction à l'algorithmique*, Dunod.

[Gondran] M. Gondran, M. Minoux, *Graphes et algorithmes*, Eyrolles.

[Nilsson] N. Nilsson, *Principes d'Intelligence artificielle*, Cépadues.