Optimal : Oui à condition que les couts sont identiques et on utilise l'exploration en largeur d'abord pour les 2 sens.

4. Procédure Heuristique de recherche avec graphe

Les méthodes de **recherche aveugle sont des méthodes exhaustives** visant à découvrir le chemin qui mène à un noeud but. Ces méthodes, malgré qu'elles arrivent à trouver un tel chemin s'il existe, ne sont pas utilisées en pratique car elles développent trop de noeuds avant d'arriver au but. Il faut trouver des alternatives plus efficaces que la recherche aveugle.

On peut utiliser certaines informations dépendantes au domaine appelées **« informations heuristiques »**. **Une heuristique est une connaissance qui sert à guider**. Les procédures de recherche utilisant de telles informations sont appelées « **procédures heuristiques** ». Certaines heuristiques réduisent considérablement l'effort de recherche mais ne garantissent pas de trouver le chemin de moindre coût. En pratique on essaye de minimiser une combinaison du coût du chemin et du coût de la recherche requise pour obtenir ce chemin. On s'intéresse à des méthodes qui minimisent la moyenne de cette combinaison.

On dit qu'une méthode de recherche 1 a plus de puissance heuristique qu'une méthode 2, si la combinaison moyenne du coût de recherche de la méthode 1 est plus basse que celle de la méthode 2.

Les coûts moyens ne sont jamais en réalité calculés, car il est difficile de décider du moyen de combiner le coût du chemin et le coût de recherche et aussi il est difficile de définir une distribution statistique sur le problème. Pour cette raison c'est à l'intuition et l'expérience qu'il appartient de décider si une méthode a plus de puissance heuristique qu'une autre.

4.1 Utilisation des fonctions d'évaluation.

L'information heuristique peut être utilisée pour ordonner les noeuds dans « Ouvert ». On peut utiliser une fonction à valeurs réelles sur les noeuds appelée ***fonction d'évaluation- (promesse d'un nœud)*** (par exemple la probabilité que le noeud se trouve sur le bon chemin).

Soit une telle fonction désignée par f et soit f(n) sa valeur au niveau du noeud n. On supposera que cette fonction donne l'estimation du coût d'un chemin optimal allant du noeud de départ d au noeud but en passant par n. Une fois cette fonction définie, on ordonne les noeuds, suivant un ordre croissant (si 2 noeuds ont la même valeur, leur classement est arbitraire en accordant la priorité au noeud but). On suppose qu'un noeud ayant une évaluation basse a plus de chance d'être sur un chemin optimal.

Exemple: Soit le problème du Taquin à 9 cases. On utilise la fonction d'évaluation: f(n)=p(n) + w(n), où p(n) est la profondeur du noeud n dans l'arbre de recherche et w(n) compte le nombre de cases mal placées dans cet état par rapport au but. Par exemple la configuration de départ suivante a une valeur f = 0 + 4 = 4 (profondeur 0 et 4 cases mal placées par rapport au but 1, 2, 6 et le 8).

2	8	3
1	6	4
7	x	5

Etat Initial

$\Rightarrow$

1	2	3
8	x	4
7	6	5

But

Les résultats avec cette fonction d'évaluation sont donnés par: (exercice)

Si la fonction est f(n)=p(n), on obtient la recherche en largeur d'abord.

Le choix de la fonction a un effet déterminant sur les résultats de la recherche. En effet, l'utilisation d'une fonction qui ne parvient pas à reconnaître certains noeuds prometteurs peut produire des chemins dont le coût n'est pas minimal. En revanche, une fonction qui surestime les résultats que promettent tous les noeuds produit un nombre trop élevé de noeuds développés (exemple recherche en largeur d'abord).

4.2 Recherche avec cout uniforme

On développe le nœud ayant le **coût le plus bas de l'état initial à ce nœud**. Elle est équivalente à largeur d'abord si le coût des actions est toujours le même.

Complétude : oui
Complexité en temps : $O(b^d)$
Complexité en espace : $O(b^d)$
Optimal : oui

4.3 Recherche avare (Gloutonne)

_ Stratégie la plus simple de "recherche meilleur-d'abord". Sa fonction heuristique **h(n) est une estimation du coût du noeud n au but**. Dans la recherche **avare** on essaye de minimiser le coût estimé pour atteindre le but. Le noeud qui semble être le plus proche du but sera étendu en priorité.

Complétude : Non, peut rester pris dans une boucle, est complet si espace de recherche fini et si vérification d'absence de boucle.
Complexité en temps : $O(b^m)$ exponentielle
Complexité en espace : $O(b^m)$ garde tous les noeuds en mémoire
Optimal : Non

Remarque : la performance de la recherche avare est fonction de la précision de h(n), avec une bonne fonction heuristique, les complexités en temps et en espace peuvent être fortement réduites.

4.4 Algorithme A*

La recherche **avare** minimise le coût estimé h(n) du nœud n au but réduisant ainsi beaucoup le coût de la recherche pour atteindre le but, mais elle n'est pas optimale et pas complète.

L'algorithme de recherche en **coût uniforme** minimise le coût g(n) depuis l'état initial au noeud n, il est optimal et complet, mais pas très efficace.

Idée : combiner les deux algorithmes et minimiser le coût total f(n) du chemin passant par le noeud n f(n) = g(n) + h(n).

Définition: Définissons la fonction d'évaluation f pour que la valeur f(n) estime la somme du coût du chemin optimal depuis le noeud initial d au noeud n, plus le coût du chemin optimal du noeud n à un noeud but (c.-à-d. f(n) est une estimation du coût du chemin optimal passant par n). Le noeud dans « Ouvert » ayant la plus petite valeur de f est donc le noeud estimé et approprié à être développé en 1er.

Notations: Soit les fonctions k, h* telles que:

$k(n_i,n_j)$ *donne le coût réel entre 2 noeuds n_i et n_j relié par un arc.*
$k(n,t_i)$ *donne le coût du chemin optimal du noeud n à un noeud but t_i*
$h^*(n)$ = *min($k(n,t_i)$) sur les t_i (la fonction h* n'est pas définie*

pour les noeuds à partir desquels le but n'est pas atteints)
C'est le cout minimal de n à un nœud but t_i
k(d,n) *le coût du chemin optimal du noeud de départ d à n.*
g*(n)=min k(d,n) *pour tous les nœuds n accessibles à partir de d.*

f*(n) =g*(n)+h*(n) la somme du coût réel du chemin optimal du noeud d au noeud n et du coût d'un chemin optimal du nœud n au but.

f(n) est donc le coût d'un chemin optimal commençant en d et passant par n et atteint un but.*

Nous souhaitons que f soit une estimation de f* soit f(n)=g(n)+h(n) ou g est une estimation de g* et h une estimation de h*. Un choix simple pour g(n) peut être le coût du chemin dans l'arbre de recherche de d à n donné en additionnant le coût des arcs traversés de d à n (chemin déjà parcouru). Pour l'estimation h(n) de h*(n), nous comptons sur l'information heuristique du domaine (par exemple similaire à la fonction w(n) du Pb du Taquin). h est appelée la fonction heuristique.

Nous appelons l'algorithme de **Recherche avec Graphe** utilisant la fonction f(n)=g(n)+h(n), ***l'Algorithme A***.

Si h est un minorant de h*(c.-à-d. h(n) ≤ h*(n) ∀n) alors l'algorithme A trouvera un chemin optimal vers un but. Lorsque l'algorithme A utilise une fonction heuristique h qui est un minorant de h*, nous l'appelons A*.

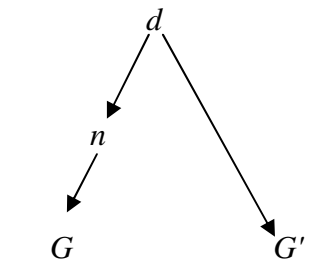
Admissibilité de A*

Nous disons qu'un algorithme de recherche (ou une heuristique) est admissible (ou minorant) si, pour tout graphe, il se termine toujours par un chemin optimal de d à un noeud but (lorsqu'un tel chemin existe) c**.-à-d. que pour tout nœud n à partir duquel un nœud but peut être atteint, h(n) <= au cout du chemin optimal de n à un noeud but.**

<u>Théorème</u> : si A* utilise une fonction heuristique admissible, c.-à-d. qui ne surestime jamais le coût réel c'est-à-dire si (∀n) 0≤ h(n) ≤ h*(n) avec h*(n) coût réel de n au but, alors A* est optimal. Une fonction heuristique admissible est toujours optimiste.

Preuve d'optimalité de A*

Supposons qu'il y ait un état but non optimal G' généré dans la liste Ouvert,
Soit n un noeud non développé sur le chemin le plus court vers un état but optimal G.



f (G') = g(G') car h(G') = 0 (car G' est un but)qui est
> g(G) car G' n'est pas optimale qui est
≥f (n) car h est admissible

et donc f (G') > f (n), donc A* ne va pas choisir G'

Propriété : Pour garantir l'optimalité, la fonction h doit être consistante. Une heuristique est **consistante** si pour tout noeud n_i et tout successeur n_{i+1} par l'action 'a', nous avons l'inégalité: $h(n_i)-h(n_{i+1)} \leq c(a)$.

Une heuristique h est monotone si $h(n_i) - h(n_{i+1}) \leq c(n_1, n_{i+1})$, c-à-d la différence entre les fonctions heuristiques de deux noeuds successifs ne doit jamais dépasser le cout nécessaire pour passer de n_i à n_{i+1}.

Une heuristique h est consistante si $h(n1) - h(n2) \leq k^*(n1, n2)$, c-à-d la différence entre les fonctions heuristiques de deux noeuds ne doit jamais dépasser le cout nécessaire pour passer de n1 à n2.

Les fonctions heuristiques consistantes sont toujours admissibles, mais les fonctions heuristiques admissibles sont très souvent, mais pas toujours, consistantes.

Propriétés A*

_ *Complétude : Oui, sauf si nombre infini de noeuds*
_ *Complexité en temps : exponentielle*
_ *Complexité en espace : garde tous les noeuds en mémoire*
_ *Optimal : Oui*

4.5 Exemples d'heuristiques pour le Taquin :

- h(N) = nombre de cases (de 1 à 8) mal placées par rapport au but
- h(N) = somme des distances (Manhattan) de chaque case (de 1 à 8) à sa position finale du but

Comparaison des algorithmes A*

Résultat : Si A1 et A2 sont 2 versions de A* telles que A2 est plus informé que A1 (c-a-d que ∀ noeud n non but, h2(n) > h1(n)), chaque noeud développé par A2 est développé par A1. Il s'ensuit que A1 développe au moins autant de noeuds que A2. (avec h2, A* développe moins ou autant de noeud qu'avec h1).

5. La Puissance Heuristique des Fonctions d'Evaluation

La sélection de la fonction heuristique est cruciale pour déterminer la puissance heuristique de l'algorithme A. Choisir h=0 (recherche en largeur d'abord) assure l'admissibilité mais s'avère inefficace. Si on prend h égale à la borne inférieur de h*, on ne développe que les quelques nœuds compatibles avec le maintien de l'admissibilité.

6. Algorithmes Voisins

Recherche Bidirectionnelle: Les méthodes de recherche vues jusqu'à présent utilisent les règles de productions soit en chaînage Avant, soit en chaînage Arrière. Une possibilité intéressante est de chercher dans les 2 directions simultanément. La recherche avance simultanément à partir du noeud de départ et d'un ensemble de noeuds but. Le processus se termine lorsque (et si) les 2 frontières de la recherche se rencontrent.

Recherche par Etapes: D'après son nom, on désire effectuer une recherche par étape. Supposons que nous ayons un problème qui même avec des heuristiques performantes, on atteint les limites de calcul, au lieu d'abandonner l'opération il serait intéressant d'élaguer le graphe pour libérer l'espace mémoire indispensable afin de pousser la recherche plus en profondeur. Il serait intéressant d'élaguer (débarrasser le graphe des chemins inutiles) puis continuer la recherche pour libérer l'espace mémoire. Même si A* est utilisé à chaque stade et que le processus total se termine en trouvant un chemin, il n'est pas garanti que le chemin soit optimal.

Limitations des successeurs: On peut se débarrasser de tous les successeurs n'ayant pas une valeur de f satisfaisante après qu'ils ont été développes. Cet élagage fait perdre au système sa complétude.

Recherche Locale Gloutonne.

L'algorithme local le plus simple s'appelle la recherche locale gloutonne. Nous commençons avec un état initial, si l'état est un état but nous arrêtons la recherche, Sinon nous générons ses successeurs et leurs valeurs heuristiques. S'il n'existe pas de successeur avec une meilleure valeur que la valeur heuristique de l'état actuel, nous pouvons plus améliorer la situation, donc nous arrêtons la recherche. Sinon, nous choisissons l'état successeur ayant la meilleure valeur heuristique, et nous continuons ainsi.
L'avantage de la recherche locale gloutonne est qu'elle a une très faible consommation en mémoire (il ne faut garder que l'état actuel en mémoire) et aussi en temps. L'inconvénient principal de la recherche locale gloutonne est son faible taux de succès qui résulte du fait que nous arrêtons la recherche si jamais nous ne pouvons plus améliorer directement la valeur heuristique.