ESIEE Paris

PR-3602

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

Auteurs :
Boris Ghidaglia
Augustin Prolongeau

Encadrant :
M.Couprie

30 avril 2018

Sujet : https ://perso.esiee.fr/ coupriem/PR3602/

Table des matières

1 Algorithme naïf

1.1 Concept

Tester toutes les solutions possibles et choisir la moins couteuse.

1.2 Questions

Combien y'a-t il de solutions possibles ?
Le sujet stipule que nous sommes dans le cas d'une matrice $N \times N$. Cela signifie que chaque agent peut être attribué à N postes différents. Cependant, à chaque fois qu'un agent est affecté à un poste, c'est une combinaison en moins à tester pour tous les autres. Ainsi, il y a $N!$ solutions possibles.

Si l'on suppose qu'une affectation peut être évaluée en une microseconde, et que l'on dispose de trois mois pour faire le calcul, quelle est la valeur maximum de N possible pour envisager d'appliquer cette méthode ?
Calculons combien de microsecondes trois mois représentent (on considèrera qu'un mois dure environ 30.5 jours) :

$$3 \text{ mois} = 3 \times 30.5 \times 24 \times 3600 \times 10^6 = 7.9056 \times 10^{12} \mu s$$

Il suffit alors de prendre la plus grande factorielle inférieure à cette valeur pour connaître notre N maximal théorique :

$$16! = 2.0922789888 \times 10^{13}$$

$$15! = 1.307674368 \times 10^{12}$$

Notre N maximum théorique est donc : 15. Si nous devons gérer une équipe de plus de 15 agents à affecter à plus de 15 postes, il nous sera impossible de calculer le résultat optimal via cet algorithme en trois mois ou moins.

2 Algorithme glouton

2.1 Concept

Il s'agit de sélectionner la valeur minimale de la matrice des coûts, d'effectuer l'affectation correspondante et de retirer le poste et l'agent qui sont concernés, puis de recommencer jusqu'à affectation de la totalité de l'effectif.

2.2 Questions

Montrez par un contre-exemple simple que l'algorithme glouton ne trouve pas toujours la solution optimale pour ce problème

Posons les matrices 3×3 suivantes : situation initiale, solution glouton et solution optimale :

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 100 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 100 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 100 \end{bmatrix}$$

On constate bien que l'algorithme glouton dévore la plus petite valeur de la matrice C_k à chaque étape k, sans se soucier des conséquences de ses actes sur ses choix futurs.

3 Algorithme A*

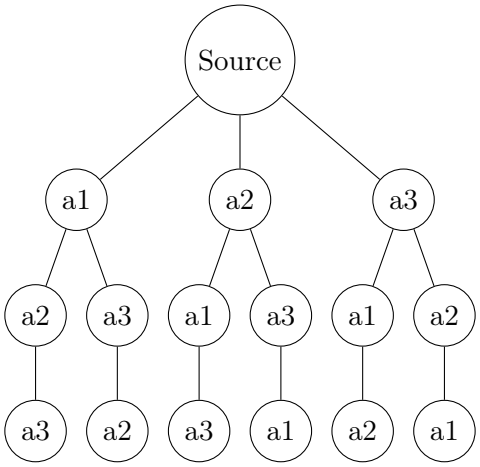
3.1 Avant d'aborder l'atelier : Test

Qu'est-ce qu'un Graphe de Résolution de Problème (GRP), relativement à un problème donné ?

Un GRP est un graphe dont les sommets représentent les états possibles d'un problème donné. On peut passer d'un état i à un état j via un arc u si une régle le permet. Cet arc se verra attribuer un coût $c(u)$. Le sommet initial représente l'état initial du problème et les sommets terminaux représentent chacun une solution possible du problème, ils se différencient principalement par le coût qui les séparent du sommet initial.

Quel GRP proposeriez-vous pour le problème de l'affectation ?

Il semble pertinent de représenter un arbre dans lequel chaque niveau $0, 1, ..., n$ illustrerait l'affectation d'un poste j appartenant à $0, 1, ..., n$, et chaque sommet serait un agent i appartenant à $0, 1, ..., n$ choisi. On ajoutera un coût c sur chaque arc. Illustration avec aX l'agent X, et sans représenter les coûts ni les numéros des postes (pour des raisons $LaTeXienne$) :



(1)

Quel est, schématiquement, le fonctionnement d'un algorithme A* ?
On pose :

- $OUVERT$: une structure de données contenant les sommets découverts et non visités. On la maintiendra ordonnée par f.
- $FERME$: une structure de données contenant les sommets visités.

L'algorithme évolue schématiquement ainsi :

- Tant que $OUVERT$ non vide
- On récupère le meilleur sommet de la liste $OUVERT$, triée sur la base des valeurs de f des sommets.
- **Si** ce sommet est un/l' objectif, FIN : on retourne le chemin vers ce sommet en remontant les prédécesseurs. Ce chemin est optimal si $h \leq h*$
- **Sinon** on le place dans $FERME$ et on ajoute ses successeurs à $OUVERT$.

Que représentent les symboles g, h et f dans l'algorithme ?
Dans l'algorithme, g est la somme des coûts entre la source et un sommet k via un chemin c. h est une heuristique. Autrement dit, une estimation du coût entre ce sommet k et un objectif n. Enfin : $f = g + h$

Quelle est la condition sur h pour que l'on parle d'algorithme A* ?
Pour que l'on parle d'algorithme A^*, on doit poser une heuristique h telle que $h \leq h^*$, où h^* est l'heuristique parfaite, l'estimation idéale.

3.2 Heuristique nulle

3.2.1 Concept

L'heuristique nulle consiste simplement à attribuer une valeur 0 aux h de tous les sommets. L'algorithme se comporte alors comme celui de Dijkstra.

3.2.2 Preuve

On pose C notre matrice de coûts, et c un coût. h_0 l'heuristique nulle, h^* l'heuristique idéale.

$$\left.\begin{array}{r} \forall c \in C, c \geq 0 \implies h^* \geq 0 \\ h_0 = 0 \end{array}\right\} \implies h_0 \leq h^* \tag{2}$$

3.3 Somme des coûts minimums restants, par ligne

3.3.1 Concept

Il s'agit de sommer les coûts minimums d'affectation des jobs non encore affectés à n'importe lequel des agents non encore affectés. Cette somme de minimums a été désignée comme étant "par ligne", car si l'on se rapporte à notre arbre, chaque ligne correspond en fait à un job.

3.3.2 Preuve

Comme nous l'avons compris grâce à l'arbre de la partie 3.1, une affectation optimale correspond au chemin de moindre coût joignant la source à une feuille. De plus, par définition de cette heuristique, on peut affirmer qu'à tout moment de l'algorithme, la valeur h estimée par l'heuristique sera inférieure ou égale (cas où les valeurs minimums restantes à chacune des lignes restantes peuvent exister dans une même combinaison d'affectations) à la valeur de h^*.

3.4 Somme des coûts minimums restants, par colonnes

3.4.1 Concept et Preuve

Même réflexion que pour l'heuristique : "Somme des coûts minimums restants, par ligne", mais en parcourant pour chacun des agents non encore affectés leurs jobs non encore affectés.

3.5 Maximum des valeurs des heuristiques par lignes et colonnes

3.5.1 Concept et intérêt

Le concept est simple : prendre la valeur maximum entre le résultat donné par l'heuristique 3.3 et 3.4. Cela est intéressant car, h^* étant l'heuristique parfaite, plus on se rapproche de sa valeur, meilleure notre estimation est, et donc plus notre algorithme sera efficace en général.

3.5.2 Preuve triviale

Posons h_1 l'heuristique 3.3, h_2 l'heuristique 3.4, h l'heuristique "Maximum des valeurs des heuristiques par lignes et colonnes" h^* l'heuristique idéale.

$$\left.\begin{array}{l} h_1 \leq h^* \\ h_2 \leq h^* \end{array}\right\} \implies max(h_1, h_2) = h \leq h^* \tag{3}$$

3.6 Minimum coefficienté

3.6.1 Concept

Contrairement à l'heuristique précédente qui est plus proche de h^* que les 3.3 et 3.4, celle-ci est plus éloignée, et donc moins efficace théoriquement. Son concept est le suivant : on prend la valeur minimum parmi celles disponibles et on la multiplie par le nombre d'affectations restant à effectuer (équivalent à la valeur que nous utilisons dans le code : le nombre d'agents disponibles).

3.6.2 Preuve

Si l'on prend la valeur v minimum parmi les valeurs disponibles et que l'on multiplie celle-ci par le nombre d'étapes p restantes, alors nous aurons nécessairement formé une valeur inférieure ou égale à h^*. En effet, h^* correspond à une somme de valeurs différentes qui s'ajoutent à chaque étape. Or v est la plus petite de toutes les valeurs que h^* pourrait être ammené à sommer. Ainsi, multiplier v par p donne nécessairement une valeur inférieure à h^*.

3.7 Performances

Dû à un problème de précision lié au calcul du temps d'exécution, nous avons créé une machine virtuelle Linux, le module python time étant plus précis sous Linux que sous Windows.

3.7.1 Matrice A

Heuristique	Exécution (s)	Noeuds visités	Ratio Visités/Total
Nulle	0.00016	7	0.777
Min par ligne	0.00027	5	0.555
Min par colonne	0.00015	6	0.666
Max(lignes, colonnes)	0.00016	5	0.555
Min coefficienté	0.00014	6	0.666

3.7.2 Matrice B

Heuristique	Exécution (s)	Noeuds visités	Ratio Visités/Total
Nulle	0.00008	8	0.888
Min par ligne	0.0001	6	0.666
Min par colonne	0.00009	4	0.444
Max(lignes, colonnes)	0.00018	4	0.444
Min coefficienté	0.00011	6	0.666

3.7.3 Matrice C

Heuristique	Exécution (s)	Noeuds visités	Ratio Visités/Total
Nulle	0.00012	13	0.393
Min par ligne	0.00030	7	0.212
Min par colonne	0.00014	5	0.151
Max(lignes, colonnes)	0.00024	5	0.151
Min coefficienté	0.00027	8	0.242

3.7.4 Matrice D

Heuristique	Exécution (s)	Noeuds visités	Ratio Visités/Total
Nulle	0.00016	17	0.515
Min par ligne	0.00017	5	0.151
Min par colonne	0.00027	9	0.272
Max(lignes, colonnes)	0.00030	5	0.151
Min coefficienté	0.00035	12	0.363

3.7.5 Matrice E

Heuristique	Exécution (s)	Noeuds visités	Ratio Visités/Total
Nulle	33.4027	471337	5.018e-06
Min par ligne	0.76576	854	9.092e-09
Min par colonne	0.03716	65	6.920e-10
Max(lignes, colonnes)	0.06729	60	6.387e-10
Min coefficienté	20.1885	66813	7.113e-07

4 Annexe

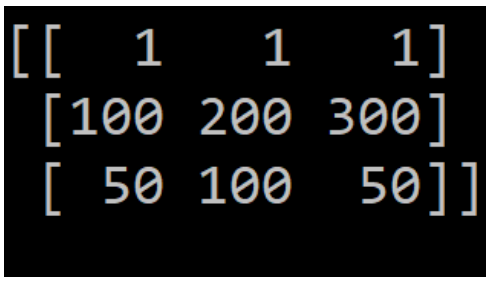


FIGURE 1 – Matrice A

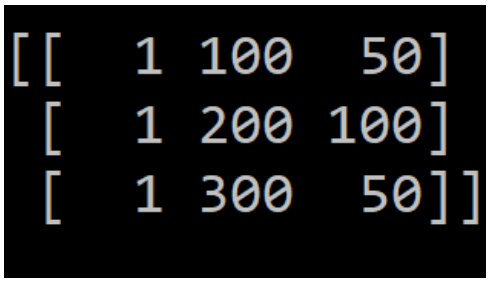


FIGURE 2 – Matrice B

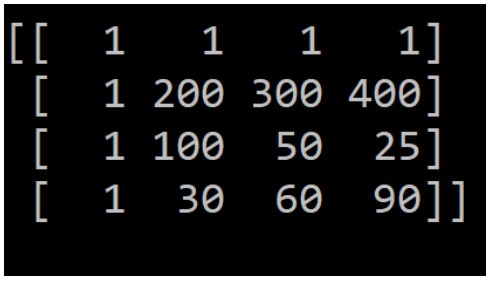


FIGURE 3 – Matrice E

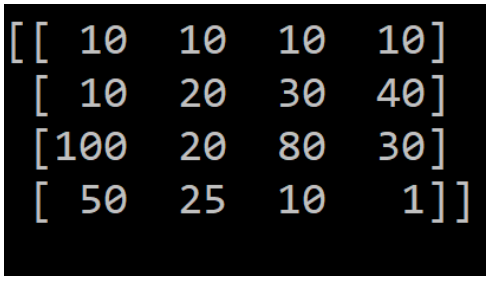


FIGURE 4 – Matrice D

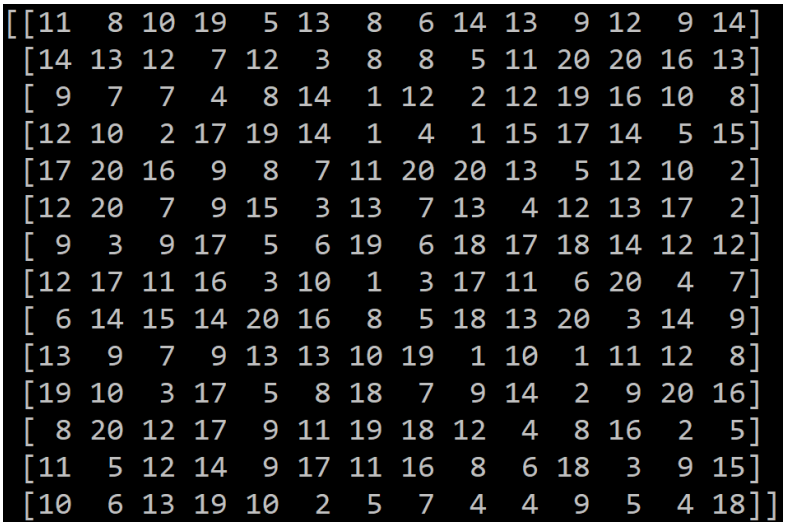


FIGURE 5 – Matrice E