Projet Métaheuristiques

Dalati – Bergerot

Nous nous intéressons dans ce projet au problème d'affectation de tâches que nous allons tout d'abord définir :

Données : N tâches, K personnes et une matrice $T_{i,j}$ du temps de la personne i pour effectuer la tâche j.

Espace de recherche : Tableau d'affectation de chaque tâche à une personne.

Objectif : Minimiser *objectif = max* $\sum T_{i,j}.\delta(j, affect[i])$

avec δ la fonction qui retourne 1 si la tâche j est affectée à la personne i sinon renvoie 0.

Pour traiter ce problème nous avons choisi de nous intéresser aux approches par construction et plus précisemment aux algorithmes Greedy, Profondeur, A* et Branch-and-Bound.

L'approche par construction consiste à partir d'une solution vide et de la construire progressivement : à chaque étape un choix est fait et la solution qui durant l'algorithme est partielle devient au final complète. Ici une solution partielle est une solution dont toutes les tâches ne sont pas affectées alors que dans une solution complète, elles le sont toutes.

Cette approche revient donc à parcourir l'arbre des solutions possibles.

Algorithme Greedy

L’approche Greedy consiste, à chaque étape, à choisir la meilleure solution partielle. Les solutions sont évaluées par rapport au temps maximum pour une personne pour effectuer les tâches qui lui sont affectées.

Greedy parcourt donc les solutions voisines de la solution courante et choisit celle ayant l’évaluation la plus basse, jusqu’à arriver à une solution complète. L’algorithme se limite donc à une recherche d’un minimum local et donc à l’exploration d’une seule branche de l’arbre des solutions.

Algorithme de parcours en profondeur

L’approche du parcours en profondeur consiste à effectuer un parcours entier en profondeur de l’arbre des solutions et à récupérer la solution complète ayant l’évaluation la plus basse.

L’algorithme ne compare entre elles que les solutions complètes, ce qui permet à la fin du parcours de l’arbre d’être sûr de récupérer la solution complète de plus basse évaluation.

Le parcours en profondeur visite l’ensemble de l’arbre des solutions et donc est assez coûteux en nombre d’itérations.

Algorithme A*

L’approche A* consiste à favoriser l’exploration des solutions les plus prometteuses. La notion de solution prometteuse est établie grâce à une heuristique, cela consiste à regarder dans le futur de la solution et à estimer la meilleure évaluation lorsque cette solution sera complète.

L’algorithme va stocker dans une liste triée les solutions voisines de la solution courante, les solutions sont alors triées grâce à la somme de leur évaluation et de la valeur de l’heuristique qui leur est associée et l’algorithme visite donc la solutions ayant la somme la plus basse, donc la plus prometteuse.

L’heuristique que nous avons utilisé affecte aux tâches encore non affectées la personne ayant le temps de réalisation le plus court.

Algorithme Branch and Bound

L’approche Branch and Bound consiste à parcourir l’arbre de solution en utilisant le principe de l’algorithme de recherche que l’on souhaite, et d’ajouter à ce parcours (qui est la partie Branch dans le nom) une notion d’évaluation et de comparaison (partie Bound dans le nom) qui permet d’élaguer des solutions qui sont dominées par une précédente solution rencontrée. On utilise des bornes qui nous permettent de savoir à quelle valeur on peut prétendre au minimum parmi les solutions visitées, ce qui nous permet de juger de l’intérêt de visiter une solution.

Pour la recherche et le parcours de l’abre des solutions, nous utilisons le parcours en largeur.

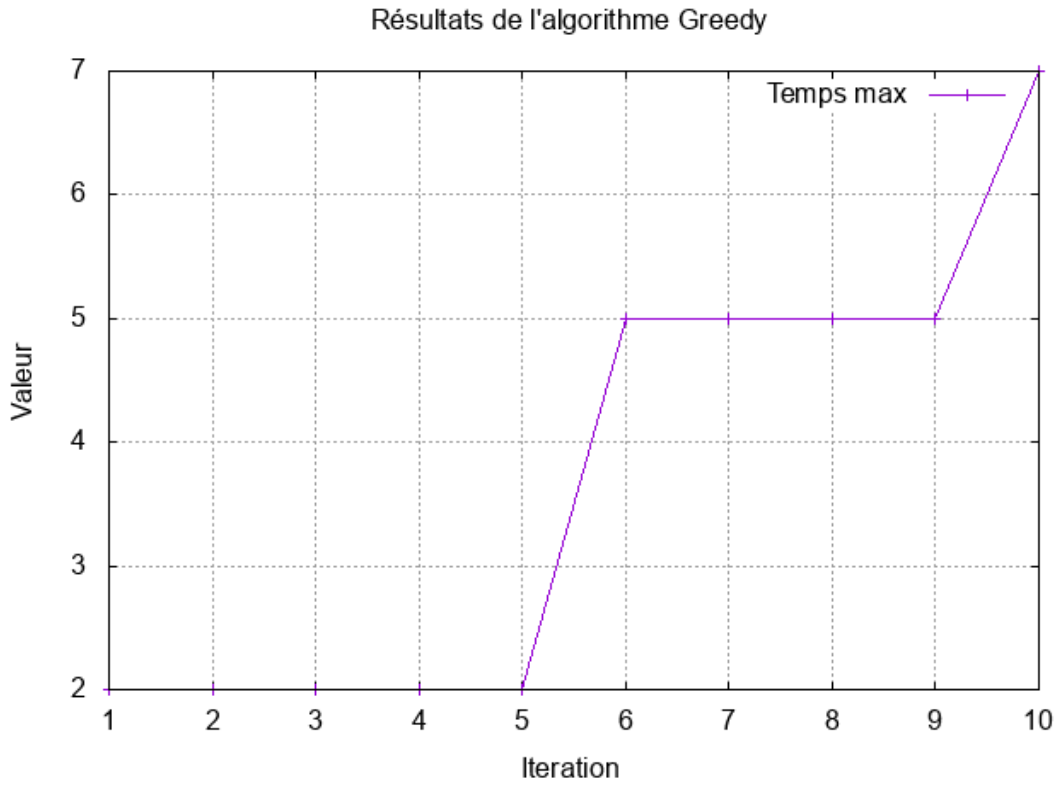
L’estimation de la valeur finale d’une solution partielle est faite grâce à une heuristique : ici nous utilisons une heuristique qui affecte aux tâches encore non affectées d’une solution la personne ayant le temps de réalisation le plus court.

Nous n’avons pas réussi à adapter le principe d’élagage d’un problème de recherche d’un maximum à notre problème ici qui est un problème de recherche d’un minimum : en effet, plus le nombre de tâches affectées augmente, plus l’évaluation augmente, et donc l’élagage va sauvegarder la valeur la plus petite comme minimum à prétendre, et donc sauvegarder les solutions où l’évaluation est la plus basse, donc celles lorsque peu de tâches sont affectées (une seule dans notre cas). Les solutions qui viendront après, avec plus de tâches affectées, ne seront donc pas visitées puisqu’elle dépassent la valeur prétendue au minimum sauvegardée.

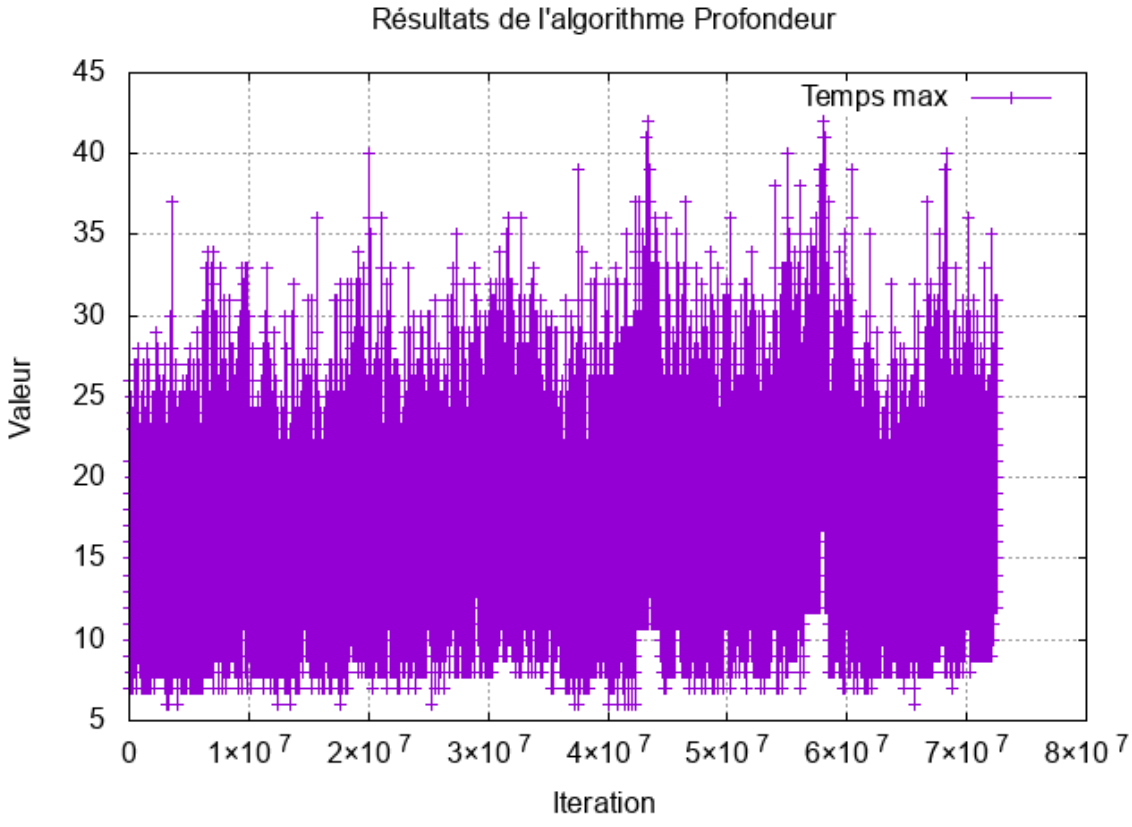
Pour résoudre ce problème, nous avons donc décidé de sauvegarder comme valeur garantie au minimum la valeur de la solution (son évaluation) sommée avec sa valeur par l’heuristique : dans ce cas, les solutions visitées dont la valeur sommée avec l’heuristique dépasse cette valeur seront élaguées, puisque sans intérêt à visiter.

Expériences

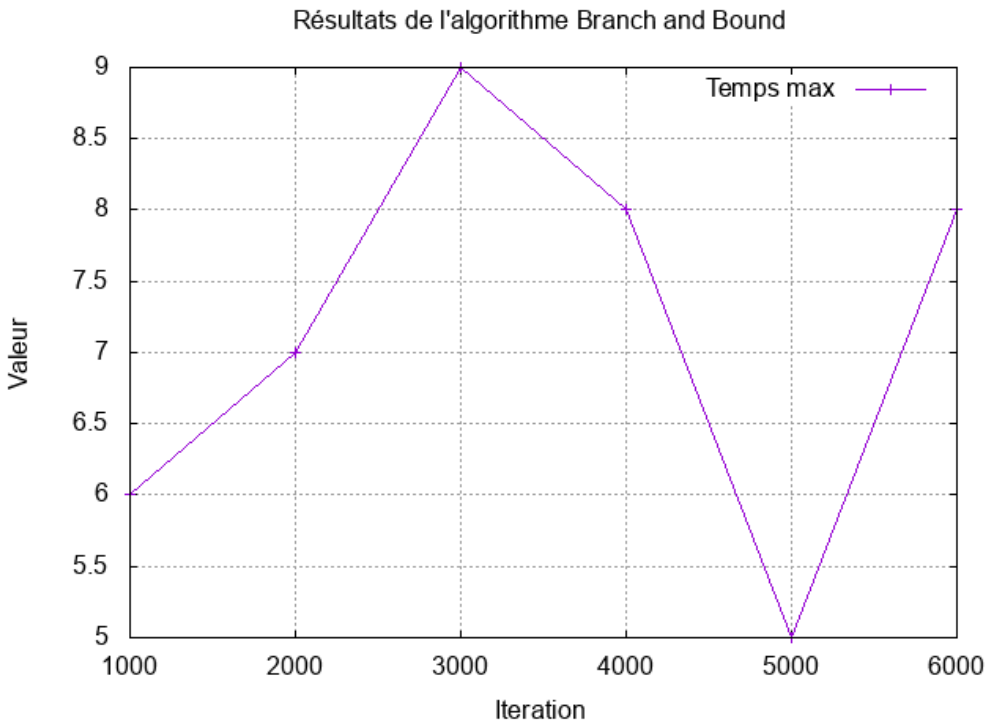
Le fichier « figures.pdf » représente les courbes pour chaque algorithme. Ces figures ont été généré par le logiciel gnuplot.



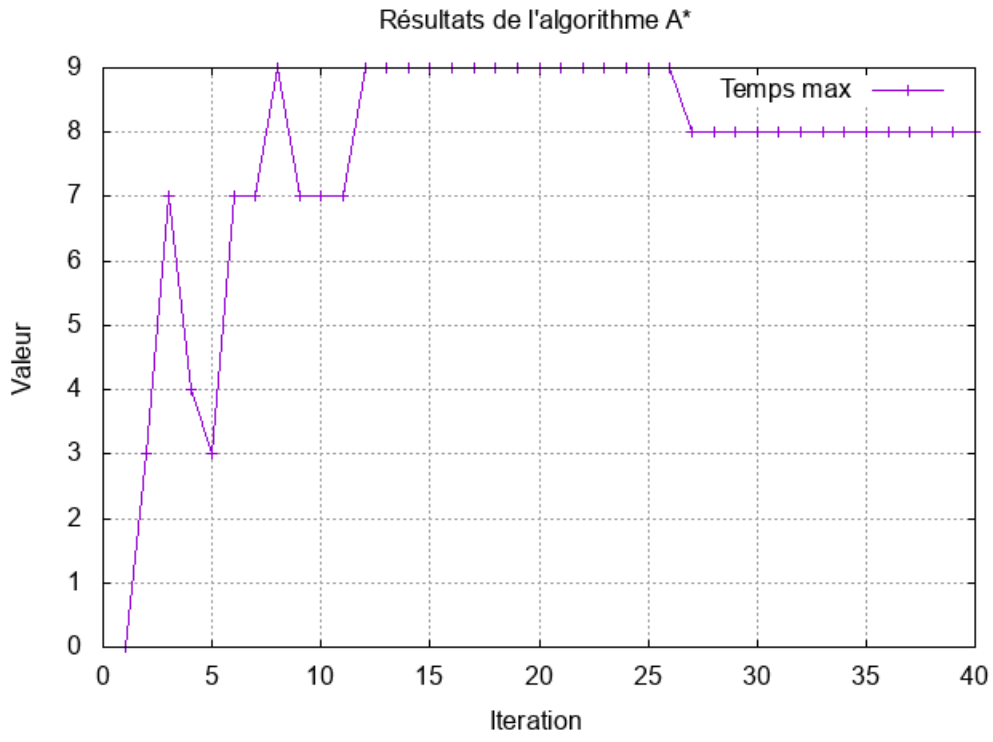
La première figure correspond à l'algorithme Greedy appliqué sur le tableau *MoinsSimple*. Nous pouvons remarquer que le temps maximum obtenu est de **7** en **10** itérations, ce qui est un assez bon résultat pour un algorithme de recherche locale, mais n'est bien sûr pas le meilleur possible, il pourra donc nous servir de comparaison par rapport aux autres algorithmes.



La deuxième figure correspond à l'algorithme Profondeur appliqué sur le tableau *MoinsSimple*. Le temps max obtenu est de **5** en **72559411** itérations. Nous pouvons voir que cet algorithme effectue beaucoup plus d'itérations que les autres puisqu'il parcourt tout l'arbre des solutions.



La troisième figure correspond à l'algorithme Branch and Bound appliqué sur le tableau *MoinsSimple*. Le temps max obtenu est de **5** en **6908** itérations. Nous remarquons bien l'efficacité de l'élagage.



La dernière figure correspond à l'algorithme A* appliqué sur le tableau *MoinsSimple*. Nous obtenons un temps max égal à **8** en **40** itérations. Cet algorithme, supposé être le meilleur, nous donne un moins bon résultat que l'algorithme Greedy. Nous pensons que ce problème vient du fait que nous n'ayons pas bien choisi notre heuristique.

Nous remarquons donc que Branch and Bound et le parcours en profondeur sont les deux algorithmes ayant les meilleurs résultats, il faut donc considérer le temps mis par chaque itération de ces deux algorithmes : une itération du parcours en profondeur est plus rapide, Branch and Bound utilise une heuristique et effectue donc le calcul de la valeur de l'heuristique pour chaque solution voisine à chaque itération. Le calcul de notre heuristique est en O(n * m) dans le pire cas, avec n le nombre de tâches non affectées dans la solution partielle et m le nombre de personnes et augmente donc considérablement le temps d'une itération. Malgré cela, nous pensons que Branch and Bound est plus performant puisqu'il élague et évite de visiter beaucoup de nœuds inutiles de l'arbre des solutions. Le choix d'une meilleure heuristique plus performante par rapport à celle utilisée ici améliorerait donc encore l'approche Branch and Bound.