Résolution du Problème du Cycle Hamiltonien à l'Aide du Backtracking : Une Analyse Comparative

BENSAID Mohamed
Auteur
mohamed.bensaid@ump.ac.ma

Pr. EL ALLALI Naoufal
Encadreur
n.elallali@ump.ac.ma

17 mars 2023

Faculté Pluridisciplinaire de Nador
Master Sciences des Données et Systèmes Intelligents

Résumé

Le problème du cycle hamiltonien est un problème NP-complet en théorie des graphes. Il s'agit de savoir si un graphe donné contient un cycle qui visite chaque sommet exactement une fois. Dans cette recherche, nous avons proposé un algorithme de backtracking pour résoudre le problème du cycle hamiltonien et avons comparé ses performances avec un algorithme de brute force et une approche de programmation parallèle. Les algorithmes ont été mis en œuvre en Java et les expériences ont été menées sur un ordinateur personnel équipé d'un processeur Intel Core i5 et de 8 Go de RAM.
Keywords : *Cycle Hamiltonian ; Java ; Backtracking ;Brute Force ;Programmation Parallèle*

I Introduction

Un cycle hamiltonien, également appelé circuit hamiltonien, cycle de Hamilton ou circuit de Hamilton, est un cycle graphique (c'est-à-dire une boucle fermée) dans un graphe qui visite chaque nœud exactement une fois (Skiena (1991)). Par convention, le graphe singleton K_1 est considéré comme hamiltonien même s'il ne possède pas de cycle hamiltonien, alors que le graphe connecté sur deux nœuds K_2 ne l'est pas.

Le cycle hamiltonien doit son nom à Sir William Rowan Hamilton, qui a conçu une énigme dans laquelle un tel chemin le long des arêtes d'un dodécaèdre était recherché (le jeu d'Icos).

Le problème du cycle hamiltonien a des applications pratiques dans les domaines de l'informatique, des transports et de la fabrication. Il peut être utilisé pour optimiser les itinéraires dans les transports, minimiser les ressources nécessaires dans la fabrication et assurer une transmission efficace des données dans les réseaux informatiques.

En général, le problème de la recherche d'un cycle hamiltonien est NP-complet (Karp (1972) ;Garey et Johnson (1990)), ce qui signifie qu'aucun algorithme en temps polynomial n'est actuellement connu pour le résoudre (c'est-à-dire qu'aucune règle particulière n'est suivie pour deviner la solution), de sorte que la seule façon connue de déterminer si un graphe général donné possède un cycle hamiltonien est d'entreprendre une recherche exhaustive, ce qui signifie que la recherche d'une solution optimale pour les grands graphes est coûteuse en termes de calcul. Par conséquent, des algorithmes d'optimisation sont nécessaires pour résoudre ce problème de manière efficace.

II L'État de l'Art

Le problème du cycle hamiltonien a été largement étudié dans la littérature et divers algorithmes d'optimisation ont été proposés pour le résoudre. La force brute est l'un des algorithmes les plus simples pour ce problème, mais il est coûteux en termes de calcul pour les grands graphes. Le backtracking est un algorithme plus efficace qui explore systématiquement l'espace des solutions. Il a été démontré que le backtracking est plus performant que la force brute pour le problème du cycle hamiltonien (Escoffier, Bonifaci, et Ausiello (2011)). Le calcul parallèle a également été utilisé pour améliorer les performances des algorithmes de backtracking pour le problème du cycle hamiltonien.

III Méthodologie

Dans cette recherche, nous avons comparé deux algorithmes d'optimisation pour le cycle hamiltonien : la force brute et le backtracking. La force brute est un algorithme simple qui examine toutes les solutions possibles, tandis que le backtracking est un algorithme plus sophistiqué qui explore systématiquement l'espace des solutions.Nous avons mis en œuvre les algorithmes en Java et mené des expériences sur un ordinateur personnel équipé d'un processeur Intel Core i5 et de 8 Go de RAM.

Backtracking

Le backtracking est une technique utilisée pour trouver des solutions à des problèmes combinatoires tels que la recherche d'un cycle hamiltonien dans un graphe. L'algorithme de backtracking permettant de trouver un cycle hamiltonien dans un graphe fonctionne comme suit :

1. Choisir un sommet quelconque comme sommet de départ.
2. Marquer le sommet choisi comme visité.
3. Si tous les sommets sont visités, vérifiez si le dernier sommet du chemin actuel est adjacent au sommet de départ. Si c'est le cas, un cycle hamiltonien est trouvé. Dans le cas contraire, il faut revenir en arrière, annuler le marquage du dernier sommet et essayer un autre chemin.
4. Si tous les sommets ne sont pas visités, choisissez un sommet non visité adjacent au dernier sommet du chemin actuel.
5. Répétez les étapes 2 à 4 jusqu'à ce que tous les sommets soient visités ou que tous les chemins soient explorés.

Algorithm 1 Recherche de cycles hamiltoniens

```
1: procedure HAMILTONIAN(k)
2:     while False do
3:         NEXTVERTEX(k)
4:         if k==n then
5:             PRINT(x[1 : n])
6:         else
7:             HAMILTONIAN(k + 1)
8:         end if
9:     end while
10: end procedure
```

Algorithm 2 Chercheur de sommets suivant

```
1: procedure NEXTVERTEX(k)
2:     while False do
3:         x[k] ←x[k]+1 mod(n + 1)
4:         if x[k]==0 then
5:             return
6:         end if
7:         if G[x[k − 1], x[k]] !=0 then
8:             for j=1 to k − 1 do
9:                 if x[j] == x[k] then
10:                    break
11:                end if
12:                if j==k then
13:                    if k > n or (k==n && (G[x[k − 1], x[k]] ! = 0) then
14:                        return
15:                    end if
16:                end if
17:            end for
18:        end if
19:    end while
20: end procedure
```

la complexité de l'algorithme est O(n !)

Exemple Pratique

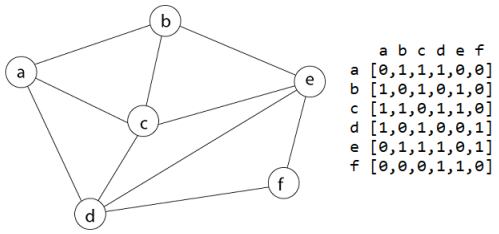

FIGURE 1 – Exemple de graph

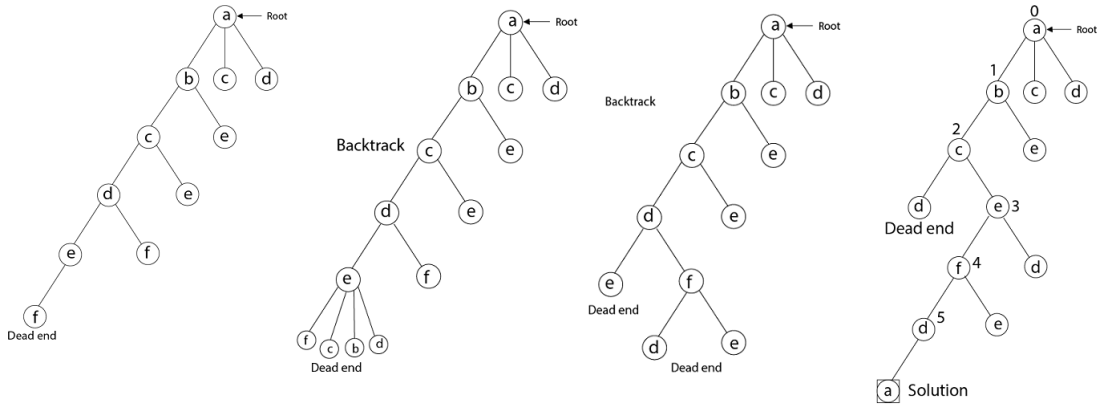

FIGURE 2 – Exemple processus de backtracking

Bruteforce

Un algorithme de force brute pour trouver un cycle hamiltonien consiste à générer tous les cycles possibles dans le graphe et à vérifier chacun d'entre eux pour voir s'il répond aux critères d'un cycle hamiltonien. L'idée de base de l'algorithme est de générer toutes les permutations possibles des sommets du graphe et de vérifier ensuite si chaque permutation forme un cycle hamiltonien. Une liste des permutations possibles est générée (à l'aide d'une méthode de backtracking). le code s'exécute en mode séquentiel donc pour accélérer le processus nous avons fait une version parallèle qui vérifie toutes les permutations simultanément. Voici un aperçu de l'algorithme de force brute :

1. Créer une liste de permutations
2. Sélectionner une permutation
3. vérifier s'il s'agit d'un cycle hamiltonien si oui l'imprimer à l'écran
4. s'il n'y a plus de permutation, arrêter sinon revenir à 2

IV Résultats

Nous avons utilisé un ensemble d'instances de test consistant en des graphes générés de manière aléatoire avec 2, 4, 8, 6 et 10 sommets. Pour chaque instance, nous avons

exécuté les algorithmes 10 fois et calculé le temps d'exécution moyen. Nous avons également comparé la qualité des solutions obtenues par les algorithmes. Le tableau ci-dessous résume les résultats obtenus.

Methode/le nombre des sommets	4	6	8	10
Backtracking	13 ms	15 ms	44 ms	170 ms
Sequential Bruteforce	12 ms	18 ms	60 ms	486 ms
Parallel Bruteforce	10 ms	13 ms	30 ms	262 ms

Pour un faible nombre de sommets, nos résultats montrent que l'algorithme de force brute dans son mode parallèle surpasse légèrement l'algorithme de backtracking en termes de durée d'exécution.en revanche, lorsque nous avons testé avec un plus grand nombre de sommets, les performances de la force brute ont commencé à chuter et le backtracking est devenu plus pertinent.pour le mode séquentiel de la force brute, la durée d'exécution est considérablement plus élevée, il ne surpasse le backtracking que pour un très petit nombre de sommets. La figure 3 montre la durée d'exécution moyenne des algorithmes pour chaque instance de test. Comme on peut le voir, l'algorithme de backtracking a la durée d'exécution la plus faible pour presque toutes les instances. L'algorithme de force brute a la durée d'exécution la plus élevée, suivi par la programmation parallèle.

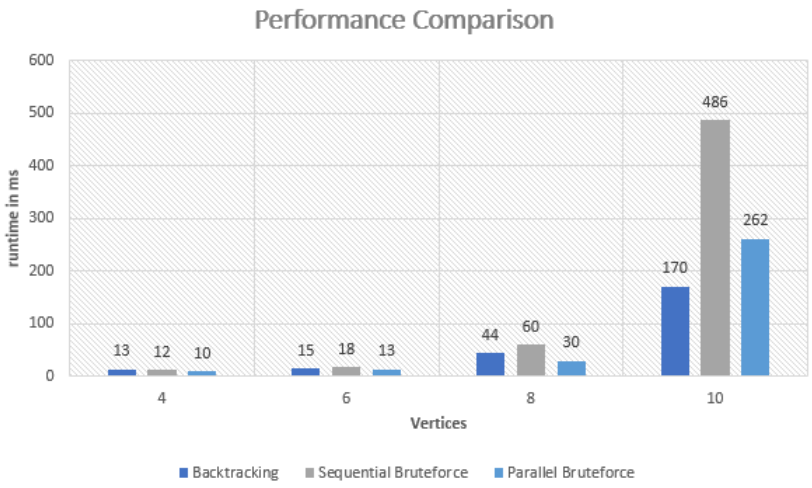

FIGURE 3 – Exemple processus de backtracking

V Discussion

Les résultats de notre étude démontrent que le Backtracking fournit une solution efficace au problème des cycles hamiltoniens, en particulier pour les grands graphes. L'algorithme de force brute n'est pas pratique pour les grands graphes en raison de la croissance exponentielle du nombre de cycles générés. L'approche Bruteforce parallèle présente une amélioration significative du temps d'exécution, ce qui en fait une option viable pour les grands graphes. Cependant, pour les très petits graphes, les frais généraux de la paralléli-

sation dépassent les avantages. Par conséquent, le choix de l'algorithme dépend de la taille du graphe et des ressources informatiques disponibles.

VI Conclusion

Dans cette étude, nous avons exploré l'utilisation du Backtracking pour résoudre le problème du cycle hamiltonien et avons comparé les résultats avec deux autres méthodes, l'algorithme séquentiel et l'algorithme parallèle Brute Force . Nos résultats montrent que le Backtracking fournit une solution efficace au problème, en particulier pour les grands graphes. L'approche de programmation parallèle montre une amélioration significative du temps d'exécution par rapport à la méthode séquentielle, ce qui en fait une option viable pour les grands graphes mais avec des ordinateurs très performants. Notre étude contribue à la recherche en cours d'algorithmes efficaces pour le problème du cycle hamiltonien et donne un aperçu des compromis entre les différentes approches.

Références

Escoffier, B., Bonifaci, V., & Ausiello, G. (2011, 02). Complexity and approximation in reoptimization.
doi: 10.1142/9781848162778_0004

Garey, M. R., & Johnson, D. S. (1990). *Computers and intractability ; a guide to the theory of np-completeness*. USA : W. H. Freeman Co.

Karp, R. M. (1972). *Reducibility among combinatorial problems* (R. E. Miller, J. W. Thatcher, & J. D. Bohlinger, Eds.). Boston, MA : Springer US. Consulté sur https://doi.org/10.1007/978-1-4684-2001-2_9 doi: 10.1007/978-1-4684-2001-2_9

Skiena, S. (1991). *Implementing discrete mathematics : combinatorics and graph theory with mathematica*. Addison-Wesley Longman Publishing Co., Inc.