Rapport : Assemblage d'un cube à partir de 6 pièces

M. Migne*A. El Ahmar†

22 septembre 2023

*Matthieu.Migne@grenoble-inp.org>
†Anas.ElAhmar@grenoble-inp.org

1 Contexte

Ce rapport détaille la démarche algorithmique adoptée pour résoudre le défi de l'assemblage d'un cube en trois dimensions à partir de six pièces découpées dans des plaques de taille 5x5 (toutes les pièces ont la même épaisseur qui est inférieure au autres dimensions donc le cube est forcément vide de l'intérieur).

2 Problématique

Ce défi soulève des difficultés liées à la disposition des pièces sur les faces du cube, à la gestion des pièces, ainsi qu'à la création de structures de données efficaces pour simuler le processus d'assemblage. L'objectif est donc de trouver un algorithme capable de déterminer la faisabilité de créer un cube en utilisant ces six pièces données en entrée.

3 Pistes de réflexion

Étant donné qu'on a que 6 pièces, nous avons envisagé l'utilisation d'un algorithme de force brute pour tenter de former le cube. Cette approche implique l'exploration de différentes combinaisons possibles de pièces, en continuant tant que nous pouvons les superposer de manière adéquate. Toutefois, la complexité théorique de ce problème est trompeuse, car sa traduction algorithmique devient complexe en raison de la dimension tridimensionnelle, qui introduit de nouvelles contraintes liées aux angles et aux orientations. Cette difficulté nous a amenés à revoir notre perspective sur le problème et à explorer une approche bidimensionnelle (par abus de langage, car les pièces ont toujours une dimension d'épaisseur). Cette approche est rendue possible grâce à la nature des pièces, qui permettent d'établir une équivalence entre le cube et son motif en deux dimensions.

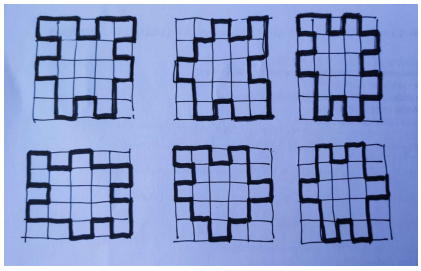

FIGURE 1 – Exemple liste de pièces

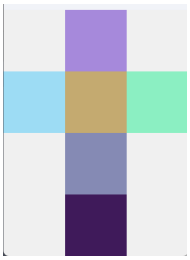

FIGURE 2 – Patron cube

4 Notre algorithme

En se basant sur l'équivalence entre cube en 3D et motif en 2D nous proposons un algorithme qui suit une approche de recherche récursive utilisant une combinaison de Depth First Search (DFS) et de Backtracking pour essayer de trouver une configuration valide pour assembler les pièces dans le cube. Cela se fait de la manière suivante :

1) Choix initial : L'algorithme commence par prendre une pièce et la place dans le cube simulé. cette pièce est la pièce centrale dans le patron du cube et considérée comme pièce de référence.

2) Recherche récursive : Ensuite, pour chaque pièce restante, il tente de trouver laquelle peut s'assembler avec un côté de la pièce déjà placée dans le cube. Si une pièce correspond, l'algorithme continue à essayer d'assembler une autre pièce qui s'emboîte avec les pièces déjà placées.

3) Backtracking : Si l'algorithme atteint un point où plusieurs pièces peuvent potentiellement s'assembler, mais l'une d'entre elles mène à une configuration non réalisable, il utilise le backtracking pour revenir en arrière et essayer les autres configurations possibles.

4) Recherche de la configuration valide : L'algorithme continue à explorer et à revenir en arrière jusqu'à ce qu'il trouve une configuration valide ou épuise toutes les possibilités.

Concernant l'implémentation, nous avons adopté une approche matricielle pour représenter à la fois le cube et les pièces. Chacune de ces entités est un objet, ce qui simplifie considérablement la manipulation de leurs matrices correspondantes.

En ce qui concerne la représentation de l'entrée, nous utilisons une liste qui stocke les informations relatives aux bords extérieurs de chaque pièce. Par défaut, nous présumons que les pièces sont pleines, car les configurations où ce n'est pas le cas sont généralement peu intéressantes et conduisent à des solutions impossibles. Cette représentation simplifiée par les bords extérieurs facilite le traitement de l'algorithme et accélère le processus de résolution.

5 Complexité de l'algorithme

Il est important de noter que la complexité du problème peut varier en fonction des dispositions initiales des pièces et de leurs interactions. Cependant on peut avoir une idée en regardant la complexité dans le pire des cas, c'est à dire on teste toutes les combinaisons possibles avec toutes les orientations possibles. Or une piéce a 8 orientations possibles et comme on a 5 pièces à tester (car on a fixé une pièce comme pièce de référence) ça fait 8^5 orientations possibles pour une seule configuration d'emplaçement des 5 pièces, et comme on a 5! configuration possibles (une simple permutation des places des 5 pièces) donc cela fait au total $8^5 * (5)!$ combinaisons à parcourir dans le pire des cas. Plus généralement, on a $8^{n-1} * (n-1)!$ combinaisons à parcourir dans le pire des cas avec n le nombre de faces du cube.

En utilisant la formule de Stirling qui donne une équivalence au factoriel on peut confirmer que dans le cas général la complexité dans le pire des cas est de l'ordre $(4n)^n$ ce qui est énorme et trés couteux et donc un algorithme inefficace dans le cas général.

.

6 Améliorations possibles

Nous avons commencé une visualisation disponible en mettant la varible debug à True dans le fichier *main.py*, mais celle-ci est très basique et donc pourrait être grandement améliorée. Concernant notre algorithme, nous pourrions sûrement discriminé dès le départ beaucoup de combinaisons possibles, cela permettrait donc de réduire grandement la complexité de notre algorithme.

7 Déroulement du travail

Étant donné que notre équipe se compose seulement de deux membres, nous avons pris la décision initiale d'explorer individuellement diverses pistes de réflexion. Chacun de nous s'est concentré sur la conception et l'implémentation d'un prototype visant à explorer les premières étapes de la piste choisie. Une fois que nous avons sélectionné l'algorithme approprié, nous avons commencé à travailler en pair programming, compte tenu de notre petite équipe, ce qui s'est avéré être une approche plus rapide et efficace pour le développement.