Université Paris Nanterre

Licence CMI-D3S

Rapport

TD2 : Le problème de la couverture par des ensembles

13 octobre 2024

Membres du groupe

Ristic Charlotte
Lafay Mathilde

Table des matières

article [utf8]inputenc [french]babel amsmath

1 Généralités

1.1 Environnement de travail

Dans le cadre de ce projet, nous avons utilisé l'éditeur de code Visual Studio Code (VS Code) avec un environnement Linux via WSL2. Par ailleurs, l'extension Live Share nous a permis de collaborer en simultané sur les fichiers, facilitant ainsi le suivi mutuel des modifications apportées au code et garantissant une cohérence dans les méthodes employées.

1.2 Structure du projet

Le projet est structuré comme suit :

- **Un fichier** `Optimal` : Ce fichier contient la résolution brute-force du problème, qui énumère toutes les solutions possibles à partir des sous-ensembles de l'ensemble principal.
- **Un fichier** `Glouton` : Un algorithme glouton qui permet de sélectionner une collection d'ensemble solution en choisissant à chaque itération, l'ensemble qui couvre le plus d'éléments parmi ceux non encore couverts.
- **Les bibliothèques** `tools.h` **et** `tools.c` : Ces fichiers intègrent des fonctions supplémentaires, notamment pour faciliter la lecture de la matrice.
- **Un fichier** `Makefile` : Ce fichier précise les règles de compilation. Pour compiler, il suffit de se placer dans le répertoire du projet, puis d'exécuter la commande `make`. Ensuite, selon l'algorithme souhaité, on exécute soit `./optimal`, soit `./glouton`. Enfin, la commande `make clean` permet de nettoyer les fichiers générés.

2 Présentation de l'algorithme optimal

2.1 Définition

Cet algorithme vise à déterminer tous les sous-ensembles possibles qui couvrent l'ensemble des éléments de l'univers U. Nous utilisons la matrice présentée dans l'énoncé pour représenter ces sous-ensembles et leurs relations.

2.2 Notre démarche

Dans un premier temps, nous avons envisagé de sommer les lignes de la matrice jusqu'à obtenir un sous-ensemble dont tous les éléments soient supérieurs ou égaux à 1. Un tel résultat indique que l'ensemble est entièrement couvert, c'est-à-dire que tous les éléments de U sont présents dans les sous-ensembles sélectionnés.

Dans un second temps, nous avons changé d'approche en cherchant à parcourir les sous-ensembles en commençant par ceux dont le premier élément est 1, puis en poursuivant jusqu'au premier 0. À ce stade, nous comparions les sous-ensembles entre eux, en cherchant un 1 dans les positions des 0, et en continuant jusqu'à obtenir une couverture complète.

Finalement, nous avons opté pour une méthode consistant à créer chaque combinaison d'ensemble possible et à vérifier si celui-ci était solution, plutôt que de chercher directement la solution. Chaque ensemble est comparé à un autre. S'il est solution, on l'affiche. Puis dans tous les cas, on Continue les comparaisons en traitant combinant ces deux ensembles afin de le traiter comme un unique ensemble. De cette manière, nous réduisons progressivement le nombre d'ensembles et avons un nombre limité de comparaisons, en ne manipulant toujours que deux sous-ensembles à la fois. Ce processus se répète jusqu'à obtenir une couverture complète.

Cette approche nous permet de traiter l'ensemble des sous-ensembles solutions comme un tout, simplifiant ainsi les comparaisons.

2.3 Explication de l'algorithme

L'algorithme optimal consiste à explorer toutes les combinaisons possibles de sous-ensembles pour trouver une couverture minimale de l'univers U. Voici les principales étapes de l'algorithme :

- Générer toutes les collections d'ensembles possibles de l'univers U.
- Pour chaque sous-ensemble, vérifier si la somme des éléments dans les sous-ensembles donne une couverture complète de l'univers.
- A chaque itération, on rappelle notre fonction de génération de combinaison, qui va garder notre collection et lui ajouter l'ensemble suivant et le tester à nouveau (et ainsi de suite...)
- Continuer jusqu'à ce que toutes les combinaisons aient été explorées.

Cet algorithme est coûteux en termes de temps d'exécution car il nécessite d'explorer toutes les combinaisons possibles (complexité exponentielle), mais il garantit de trouver la solution optimale.

2.4 Exemple d'exécution

Prenons un exemple où l'univers U est constitué de 7 éléments :

$$U = \{1, 2, 3, 4, 5, 6, 7\}$$

et les sous-ensembles disponibles sont :

$$S = \{A, B, C, D, E, F\}$$

avec les définitions suivantes :
A = {1, 4, 7}
B = {1, 4}
C = {4, 5, 7}
D = {3, 5, 6}
E = {2, 3, 6, 7}
F = {2, 7}

L'algorithme optimal va explorer toutes les combinaisons possibles de sous-ensembles pour trouver une couverture complète de U.

```
nombre d'éléments : 7, nombre d'ensembles : 6
Matrice chargée :
1 0 0 1 0 0 1
1 0 0 1 0 0 0
0 0 0 1 1 0 1
0 0 1 0 1 1 0
0 1 1 0 0 1 1
0 1 0 0 0 0 1
La sous-collection {A, B, C, D, E} est une couverture optimale
La sous-collection {A, B, C, D, E, F} est une couverture optimale
La sous-collection {A, B, C, D, F} est une couverture optimale
La sous-collection {A, B, C, E} est une couverture optimale
La sous-collection {A, B, C, E, F} est une couverture optimale
La sous-collection {A, B, D, E} est une couverture optimale
La sous-collection {A, B, D, E, F} est une couverture optimale
La sous-collection {A, B, D, F} est une couverture optimale
La sous-collection {A, C, D, E} est une couverture optimale
La sous-collection {A, C, D, E, F} est une couverture optimale
La sous-collection {A, C, D, F} est une couverture optimale
La sous-collection {A, C, E} est une couverture optimale
La sous-collection {A, C, E, F} est une couverture optimale
La sous-collection {A, D, E} est une couverture optimale
La sous-collection {A, D, E, F} est une couverture optimale
La sous-collection {A, D, F} est une couverture optimale
La sous-collection {B, C, D, E} est une couverture optimale
La sous-collection {B, C, D, E, F} est une couverture optimale
La sous-collection {B, C, D, F} est une couverture optimale
```

FIGURE 1 – Exécution brute force

3 Présentation de l'algorithme glouton

3.1 Définition

L'algorithme glouton vise à simplifier le problème en sélectionnant, à chaque étape, une solution optimale locale, ici le sous-ensemble qui maximise le nombre d'éléments non encore couverts.

3.2 Notre démarche

Nous avons mis en place une boucle qui identifie, à chaque itération, le sous-ensemble maximisant le nombre d'éléments non couverts. Pour ce faire, nous avons utilisé une fonction permettant de compter les éléments non encore couverts. On sélectionne celui avec le maximum d'éléments. Une fois un sous-ensemble sélectionné, il est ajouté à la solution et ses éléments sont considérés comme couverts.

Ce processus est répété jusqu'à ce que tous les éléments de l'univers U soient couverts.

3.3 Explication de l'algorithme

L'algorithme glouton fonctionne en suivant les étapes suivantes :

- Initialiser un ensemble de couverture vide.
- Tant que tous les éléments de U ne sont pas couverts :
  - Identifier le sous-ensemble avec le plus grand nombre d'éléments non encore couverts.
  - Ajouter ce sous-ensemble à la couverture.
  - Marquer les éléments de ce sous-ensemble comme couverts.
- Répéter jusqu'à ce que tous les éléments soient couverts.

Bien que cet algorithme ne garantisse pas de trouver la solution optimale, il est souvent plus rapide que l'algorithme optimal.

3.4 Exemple d'exécution

Prenons le même univers et les mêmes ensembles que dans la partie 2.4 :

```
nombre d'éléments : 7, nombre d'ensembles : 6
Matrice chargée :
1 0 0 1 0 0 1
1 0 0 1 0 0 0
0 0 0 1 1 0 1
0 0 1 0 1 1 0
0 1 1 0 0 1 1
0 1 0 0 0 0 1
initialisation tableaux et variables
tour 0
nouveau meilleur ensemble : A avec 3 elements
nouveau meilleur ensemble : E avec 4 elements
Ensemble avec le max d'éléments manquants : E
tour 1
nouveau meilleur ensemble : A avec 2 elements
Ensemble avec le max d'éléments manquants : A
tour 2
nouveau meilleur ensemble : C avec 1 elements
Ensemble avec le max d'éléments manquants : C
La sous-collection {E, A, C} est une couverture optimale
```

FIGURE 2 – Exécution algorithme glouton