

École Nationale Supérieure de Techniques Avancées -
ENSTA Paris

5OD21 - Optimisation Discrète

Projet

Partition d'un graphe en sous-ensembles connexes et équilibrés

Élaboré par :

BACCAR BEDIS
ZARGUI RAYEN

Encadré par :

MME SOUROUR ELLOUMI

3^{ème} Année SOD



GUROBI
OPTIMIZATION

Année universitaire : 2025/2026

Table des matières

Introduction	2
1.1 Contexte Général et Motivation	2
1.2 Importance et Applications Pratiques	2
1.3 Complexité et Enjeux Algorithmiques	2
Définition Formelle du Problème	3
2.1 Notations Préliminaires	3
2.2 Le Problème de la k-Partition Connexe Équilibrée (BCPk)	3
Formulations Basées sur les Flux (Flow-based)	4
3.1 Principe : La Connexité par l'Arborescence de Flux	4
3.2 La Formulation $F_k(G, w)$ (Modèle à k Sources)	4
Formulations Basées sur les Coupes (Cut-based)	5
4.1 Principe de Modélisation de la Connexité	5
4.2 La Formulation ILP $C_k(G, w)$	5
4.3 La Nécessité d'une Approche Branch-and-Cut	5
Améliorations de la Formulation par Coupes	6
5.1 Rappel de la Formulation Mathématique	6
5.2 Contraction de Sommets avant Séparation	6
5.3 Lifting du Séparateur Minimal	7
5.4 Restriction des Paires Candidates	7
5.5 Ajout d'Inégalités de Couverture	7
Implémentation et Analyse des Résultats	8
6.1 Environnement d'Implémentation	8
6.2 Implémentation des Formulations	8
6.2.1 Question 1 : Formulation Compacte par Flux (F_k)	8
6.2.2 Question 2 : Formulation Exponentielle par Coupes (C_k)	8
6.2.3 Question 3 : Implémentation des Améliorations	9
6.3 Tableau Comparatif des Formulations	9
6.4 Protocole Expérimental et Résultats	9
6.4.1 Protocole de Test	9
6.4.2 Résultats Expérimentaux	10
Conclusion Générale	13

Introduction

1.1 Contexte Général et Motivation

Le partitionnement de graphes constitue l'une des pierres angulaires de l'optimisation combinatoire et de l'informatique théorique, offrant un cadre formel pour une multitude de problèmes de division et d'assignation. Au sein de ce vaste domaine, le problème de la k -partition connexe équilibrée, désigné par l'acronyme BCPk (de l'anglais *Balanced Connected k -Partition*), se distingue par la richesse de sa structure et la pertinence de ses applications. Il s'attaque à la tâche complexe de diviser l'ensemble des sommets d'un graphe en k sous-ensembles, ou classes, tout en respectant simultanément deux contraintes fondamentales et souvent antagonistes.

Le premier défi est de nature topologique : chaque classe de la partition doit induire un sous-graphe connexe. Cette exigence de connexité impose une contrainte de contiguïté spatiale ou logique. Le second défi est d'ordre distributif : les partitions doivent être "équilibrées" selon une métrique définie, généralement liée au poids total des sommets qu'elles contiennent. L'objectif est de répartir la "valeur" ou la "charge" totale du graphe de la manière la plus équitable possible entre les k classes. La gestion simultanée de ces deux contraintes fait du BCPk un problème d'une grande complexité théorique et d'un intérêt pratique considérable [miyazawa2021].

1.2 Importance et Applications Pratiques

La pertinence du BCPk ne se limite pas à la sphère académique ; il modélise avec une remarquable fidélité de nombreux problèmes concrets issus de domaines variés [miyazawa2021].

Dans le domaine du **traitement d'images**, par exemple, le BCPk permet de segmenter une image en k régions distinctes (pixels connexes) de taille ou de contenu visuel similaire.

En **robotique et en logistique**, le problème se pose lors de l'assignation de tâches à une flotte de k robots. Une solution au BCPk permet de délimiter k zones de travail connexes, assurant que chaque robot opère dans un secteur d'un seul tenant, tout en équilibrant la charge de travail [miyazawa2021].

Une application particulièrement illustrative est la **démarcation des zones de patrouille de police**. Le graphe représente le réseau routier d'une ville, et le poids de chaque sommet est proportionnel au taux de criminalité. L'objectif est de diviser la carte en k secteurs de patrouille connexes tout en assurant une répartition équitable de la charge de travail (risque criminel) [miyazawa2021].

1.3 Complexité et Enjeux Algorithmiques

La puissance expressive du BCPk s'accompagne d'une grande complexité algorithmique. Le problème est reconnu comme étant NP-difficile, même dans des cas très restreints. Par exemple, la version pondérée pour $k = 2$ (BCP_2) est déjà NP-difficile sur des graphes aussi structurés que les grilles [miyazawa2021].

De plus, pour chaque $k \geq 2$, le BCPk est NP-difficile au sens fort, même sur des graphes k -connexes. Une conséquence directe est que le problème n'admet pas de schéma d'approximation entièrement polynomial en temps (FPTAS), à moins que $P=NP$.

Définition Formelle du Problème

2.1 Notations Préliminaires

Soit $G = (V, E)$ un graphe non orienté, où V est l'ensemble des sommets et E est l'ensemble des arêtes. Nous notons $n = |V|$ et $m = |E|$. Pour un entier positif k , le symbole $[k]$ désigne l'ensemble $\{1, 2, \dots, k\}$ [miyazawa2021].

Une fonction de poids non négative $w : V \rightarrow \mathbb{Q}_{\geq 0}$ est associée aux sommets. Pour tout $V' \subseteq V$, le poids total est $w(V') = \sum_{v \in V'} w(v)$ [miyazawa2021].

Une **k-partition** de V est une collection de k sous-ensembles non vides $\{V_i\}_{i \in [k]}$ telle que :

1. $\bigcup_{i=1}^k V_i = V$.
2. $V_i \cap V_j = \emptyset$ pour tous $i, j \in [k]$ avec $i \neq j$.

Une k-partition est dite **connexe** si, pour chaque classe V_i , le sous-graphe $G[V_i]$ est connexe [miyazawa2021].

2.2 Le Problème de la k-Partition Connexe Équilibrée (BCPk)

Le problème est formellement énoncé comme suit [miyazawa2021] :

- **INSTANCE** : Un graphe connexe $G = (V, E)$ et une fonction de poids $w : V \rightarrow \mathbb{Q}_{\geq 0}$.
- **À TROUVER** : Une k-partition connexe $\{V_i\}_{i \in [k]}$ de G .
- **OBJECTIF** : Maximiser la valeur $\min_{i \in [k]} \{w(V_i)\}$.

Cet objectif de type "max-min" vise à trouver la partition la plus équitable possible en maximisant le poids de la classe la plus "légère".

Formulations Basées sur les Flux (Flow-based)

3.1 Principe : La Connexité par l'Arborescence de Flux

Cette approche modélise la connexité d'une manière entièrement différente. L'idée est de forcer la création de k arborescences de flux disjointes, chacune correspondant à une classe de la partition. Un flux est envoyé depuis une racine (source) propre à chaque classe et est distribué à tous les autres nœuds de cette classe.

L'avantage majeur est que ces formulations sont **compactes**, c'est-à-dire qu'elles ont un nombre polynomial de variables et de contraintes. Elles peuvent donc être résolues directement par un solveur MILP standard (comme Gurobi ou SCIP) sans nécessiter de procédure de séparation complexe.

3.2 La Formulation $F_k(G, w)$ (Modèle à k Sources)

On construit un digraphe auxiliaire D en ajoutant k sources $\{s_1, \dots, s_k\}$ au graphe original G . Pour chaque sommet $v \in V$, on ajoute k arcs (s_i, v) pour $i \in [k]$. Pour chaque arête non orientée $\{u, v\} \in E$, on ajoute deux arcs orientés (u, v) et (v, u) [miyazawa2021].

Variables :

- $f_a \in \mathbb{R}_{\geq 0}$: quantité de flux sur l'arc a .
- $y_a \in \{0, 1\}$: 1 si l'arc a est utilisé (ouvert), 0 sinon.

Formulation $F_k(G, w)$ [miyazawa2021] :

$$\max \quad f(\delta^+(s_1)) \quad (3.2.1)$$

$$\text{s.t.} \quad f(\delta^+(s_i)) \leq f(\delta^+(s_{i+1})) \quad \forall i \in [k-1] \quad (3.2.2)$$

$$f(\delta^-(v)) - f(\delta^+(v)) = w(v) \quad \forall v \in V \quad (3.2.3)$$

$$f_a \leq M \cdot y_a \quad \forall a \in A(D) \quad (3.2.4)$$

$$y(\delta^+(s_i)) \leq 1 \quad \forall i \in [k] \quad (3.2.5)$$

$$y(\delta^-(v)) \leq 1 \quad \forall v \in V \quad (3.2.6)$$

$$f_a \in \mathbb{R}_{\geq 0}, y_a \in \{0, 1\} \quad \forall a \in A(D) \quad (3.2.7)$$

Où $f(\delta^+(v))$ (resp. $f(\delta^-(v))$) est le flux total sortant de (resp. entrant dans) v , et M est une grande constante (ex : $M = w(V)$).

- (3.2.1) et (3.2.2) : maximisent le flux de la source 1, qui est contrainte d'être la plus petite.
- (3.2.3) : Contrainte de conservation du flux. Chaque sommet v "consomme" une quantité de flux égale à son poids $w(v)$.
- (3.2.4) : Le flux ne peut circuler que sur des arcs ouverts.
- (3.2.5) : Chaque source ne peut envoyer du flux qu'à une seule destination (qui deviendra la racine de son arborescence).
- (3.2.6) : Chaque sommet $v \in V$ ne peut recevoir de flux que d'une seule source (soit d'un s_i , soit d'un autre $u \in V$).

Ensemble, les contraintes (3.2.5) et (3.2.6) forcent la structure des arcs actifs ($y_a = 1$) à former k arborescences disjointes, garantissant ainsi à la fois la partition et la connexité de chaque classe.

Formulations Basées sur les Coupes (Cut-based)

4.1 Principe de Modélisation de la Connexité

L'idée fondamentale est qu'un ensemble de sommets V_i est connexe si et seulement si, pour toute paire de sommets $u, v \in V_i$, il n'existe aucun sous-ensemble de sommets $S \subseteq V \setminus V_i$ qui les sépare.

Plus formellement, soient u et v deux sommets non adjacents. Un sous-ensemble $S \subseteq V \setminus \{u, v\}$ est un **(u,v)-séparateur** si u et v sont dans des composantes connexes différentes de $G - S$. On note $\Gamma(u, v)$ la collection de tous les (u, v) -séparateurs minimaux [miyazawa2021].

4.2 La Formulation ILP $C_k(G, w)$

Variables de décision : $x_{v,i} = 1$ si le sommet v est assigné à la classe i , et 0 sinon.

Formulation $C_k(G, w)$ [miyazawa2021] :

$$\max \sum_{v \in V} w(v)x_{v,1} \quad (4.2.1)$$

$$\text{s.t.} \quad \sum_{v \in V} w(v)x_{v,i} \leq \sum_{v \in V} w(v)x_{v,i+1} \quad \forall i \in [k-1] \quad (4.2.2)$$

$$\sum_{i \in [k]} x_{v,i} \leq 1 \quad \forall v \in V \quad (4.2.3)$$

$$x_{u,i} + x_{v,i} - \sum_{z \in S} x_{z,i} \leq 1 \quad \forall uv \notin E, S \in \Gamma(u, v), i \in [k] \quad (4.2.4)$$

$$x_{v,i} \in \{0, 1\} \quad \forall v \in V, i \in [k] \quad (4.2.5)$$

Les contraintes (4.2.2) ordonnent les classes par poids, permettant à l'objectif de maximiser le poids de la classe 1 (la plus légère). Les contraintes (4.2.3) assurent que chaque sommet est assigné à au plus une classe. Les contraintes (4.2.4) sont les contraintes de connexité : si u et v sont dans la classe i , alors au moins un sommet z de tout (u, v) -séparateur S doit aussi être dans la classe i .

4.3 La Nécessité d'une Approche Branch-and-Cut

Le nombre de contraintes de type (4.2.4) peut être exponentiel, car un graphe peut contenir un nombre exponentiel de séparateurs minimaux. Il est donc impossible de les générer toutes *a priori*.

L'approche **Branch-and-Cut** résout ce problème. On commence par résoudre une relaxation linéaire du modèle ne contenant que les contraintes (4.2.2), (4.2.3) et (4.2.5). Ensuite, on ajoute itérativement les contraintes de connexité (4.2.4) qui sont violées par la solution courante (qu'elle soit fractionnaire ou entière). Ces contraintes violées sont appelées "coupes".

L'efficacité de cette méthode repose sur l'existence d'un **algorithme de séparation** efficace, qui, étant donné une solution \bar{x} , peut trouver une contrainte de type (4.2.4) violée, ou prouver qu'il n'en existe pas. Pour ces contraintes, un tel algorithme existe et s'exécute en temps polynomial via des calculs de flot maximum / coupe minimum [miyazawa2021].

Améliorations de la Formulation par Coupes

Introduction et Contexte

La formulation par coupes $C_k(G, w)$ permet de résoudre le problème de partitionnement d'un graphe $G = (V, E)$ en k classes connexes et équilibrées. Cette approche repose sur l'ajout progressif de contraintes de connectivité de la forme :

$$x_{u,i} + x_{v,i} - \sum_{z \in S} x_{z,i} \leq 1, \quad \forall i \in [k], \forall (u, v) \in V^2 \text{ non adjacents, } S \text{ séparant } u, v. \quad (5.0.1)$$

Cependant, l'ensemble de ces contraintes est exponentiel, rendant leur ajout explicite impossible. La méthode **Branch-and-Cut** pallie cette difficulté en générant dynamiquement les contraintes violées à partir d'une solution fractionnaire courante. La qualité et la rapidité de cette procédure dépendent alors de deux éléments :

- la **vitesse** de détection des coupes violées (procédure de séparation) ;
- la **force** des inégalités ajoutées (leur capacité à resserrer la relaxation linéaire).

L'article de référence [miyazawa2021] propose plusieurs stratégies pour améliorer ces deux aspects. Dans ce chapitre, nous présentons les principales améliorations retenues et intégrées dans notre implémentation Julia du modèle BCP_k.

5.1 Rappel de la Formulation Mathématique

La formulation $C_k(G, w)$ introduit pour chaque sommet $v \in V$ et chaque classe $i \in [k]$ une variable binaire :

$$x_{v,i} = \begin{cases} 1, & \text{si le sommet } v \text{ appartient à la classe } i, \\ 0, & \text{sinon.} \end{cases}$$

Le modèle complet s'écrit :

$$\max \sum_{v \in V} w(v) x_{v,1} \quad (5.1.1)$$

$$\text{s.c.} \quad \sum_{v \in V} w(v) x_{v,i} \leq \sum_{v \in V} w(v) x_{v,i+1}, \quad i = 1, \dots, k-1, \quad (5.1.2)$$

$$\sum_{i=1}^k x_{v,i} \leq 1, \quad \forall v \in V, \quad (5.1.3)$$

$$x_{u,i} + x_{v,i} - \sum_{z \in S} x_{z,i} \leq 1, \quad \forall i, \forall (u, v) \text{ non adjacents, } S \text{ séparant } u, v, \quad (5.1.4)$$

$$x_{v,i} \in \{0, 1\}, \quad \forall v, i. \quad (5.1.5)$$

La contrainte (5.0.1) garantit la **connexité interne** de chaque classe i : deux sommets d'une même classe doivent être reliés par un chemin composé uniquement de sommets appartenant à cette classe.

5.2 Contraction de Sommets avant Séparation

Lors du processus de séparation, le calcul du flot maximum s'effectue sur un graphe auxiliaire orienté D_i . Sa taille croît linéairement avec celle du graphe original G , ce qui en fait la principale source de complexité.

Comme le suggère [miyazawa2021], de nombreuses variables $x_{v,i}$ prennent des valeurs proches de 0 ou 1 dans les solutions fractionnaires. Les sommets tels que $\tilde{x}_{v,i} \approx 1$ peuvent être

considérés comme définitivement affectés à la classe i . L'idée est alors de contracter ces sommets dans le graphe de séparation :

- les nœuds v avec $\tilde{x}_{v,i} > 1 - \varepsilon$ sont fusionnés dans un nœud unique ;
- les arcs de capacité infinie adjacents sont mis à jour pour préserver la structure du graphe.

Cette **contraction de nœuds** réduit significativement la taille du graphe D_i , diminuant le coût du calcul de flot maximum sans compromettre la validité des coupes générées.

5.3 Lifting du Séparateur Minimal

Le flot maximum entre deux sommets u et v fournit un ensemble séparateur S de capacité minimale, mais pas nécessairement minimal en cardinalité. Un grand ensemble S conduit à des coupes faibles.

L'article propose une étape de **lifting** consistant à retirer de S les sommets non essentiels à la séparation :

$$S' = \{z \in S \mid N(z) \cap H_u \neq \emptyset \text{ et } N(z) \cap H_v \neq \emptyset\},$$

où H_u et H_v sont les composantes de $G \setminus S$ contenant respectivement u et v . Cette opération renforce les inégalités de connectivité et améliore la convergence du Branch-and-Cut en réduisant le nombre de coupes redondantes.

5.4 Restriction des Paires Candidates

La séparation exacte nécessiterait d'évaluer toutes les paires (u, v) non adjacentes de G , soit $\mathcal{O}(n^2)$ tests par classe i . Pour limiter ce coût, l'article recommande de ne traiter que les paires les plus susceptibles de violer la contrainte :

$$x_{u,i} + x_{v,i} > 1 + \epsilon \quad \text{et} \quad (0 < x_{u,i} < 1 \text{ ou } 0 < x_{v,i} < 1).$$

Cette heuristique, appelée **séparation restreinte**, réduit le nombre de calculs de flot tout en conservant les coupes réellement pertinentes. Elle a été intégrée dans notre implémentation via un paramètre optionnel `restrict_pairs`.

5.5 Ajout d'Inégalités de Couverture

Enfin, la formulation peut être renforcée par les **inégalités de couverture** issues des contraintes d'équilibrage des poids [miyazawa2021]. Pour chaque classe $i \in [k - 1]$, on a :

$$\sum_{v \in V} w(v)x_{v,i} \leq \frac{w(G)}{k - i + 1}. \quad (5.5.1)$$

Cette borne linéaire, analogue à une contrainte de type sac à dos (knapsack), limite le poids maximal d'une classe i . Des versions *liftées* de ces inégalités peuvent encore renforcer le polyèdre de relaxation.

Implémentation et Analyse des Résultats

Ce chapitre présente l'implémentation des différentes formulations du problème BCP_k ainsi que l'analyse des résultats expérimentaux obtenus, conformément aux trois questions du projet.

6.1 Environnement d'Implémentation

L'implémentation a été réalisée en **Julia** (v1.9+) en utilisant le framework **JuMP.jl** pour la modélisation mathématique. Le solveur **Gurobi** a été choisi pour ses performances en programmation linéaire en nombres entiers (MILP) et sa prise en charge native des *callbacks*. Les bibliothèques **Graphs.jl** et **GraphRecipes.jl** ont été utilisées pour la manipulation et la visualisation des graphes.

6.2 Implémentation des Formulations

Trois niveaux de formulation et d'améliorations ont été développés, correspondant aux trois questions du projet.

6.2.1 Question 1 : Formulation Compacte par Flux (F_k)

La formulation compacte $F_k(G, W)$ (Section 4 de l'article [miyazawa2021]) a été implémentée dans la fonction `solve_BCPk_flow`.

- **Principe** : Cette approche modélise explicitement la **connexité** via des variables de flux sur un graphe orienté augmenté D comprenant les sommets du graphe G et k sources supplémentaires s_1, \dots, s_k .
- **Variables** :
 - $f_a \geq 0$: quantité de flux envoyée sur l'arc $a \in A(D)$;
 - $y_a \in \{0, 1\}$: activation de l'arc a dans la solution.
- **Contraintes** : Le modèle intègre les contraintes (3.2.2) à (3.2.6) :
 1. (3.2.2) Ordre des flux sortants : équilibre entre les poids des classes ;
 2. (3.2.3) Bilan de flux : chaque nœud v « consomme » un flux égal à son poids $w(v)$;
 3. (3.2.4) Contrainte Big-M : $f_a \leq w(G) y_a$;
 4. (3.2.5) et (3.2.6) Contraintes de topologie : chaque source s_i émet au plus un flux, et chaque v reçoit au plus un flux.
- **Résolution** : Le modèle complet est transmis au solveur sans callbacks. La solution obtenue est ensuite traduite en k sous-graphes connexes correspondant aux partitions.

6.2.2 Question 2 : Formulation Exponentielle par Coupes (C_k)

La formulation exponentielle $C_k(G, W)$ (Section 2 de l'article) a été implémentée dans la fonction `solve_BCPk_cutbased_improved` à l'aide d'une approche **Branch-and-Cut**.

- **Principe** : La connexité n'est pas imposée au départ. Le modèle de base contient uniquement les contraintes d'assignation, et les contraintes de connexité sont ajoutées dynamiquement pendant la recherche.
- **Variables** : Variables binaires $x_{v,i}$ indiquant si le sommet v appartient à la classe i .
- **Procédure de séparation** : Implémentée via un `MOI.LazyConstraintCallback` :
 1. À chaque solution entière trouvée, le callback identifie les composantes connexes dans chaque classe V_i ;

2. Si une classe n'est pas connexe, une coupe de connexité est ajoutée :

$$\sum_{v \in C} x_{v,i} \leq |C| - 1 + \sum_{w \in N(C)} x_{w,i}$$

3. Cette séparation garantit la génération dynamique des contraintes (3) de l'article.

6.2.3 Question 3 : Implémentation des Améliorations

Les améliorations proposées dans la Section 5 de l'article ont été intégrées dans la formulation C_k (Q2).

- (a) **Cover et Lifted Cover Inequalities (Section 5.1)** : L'activation des options `use_cover` et `use_lifted_cover` permet d'ajouter des inégalités de couverture avant la résolution. Ces coupes visent à renforcer la relaxation linéaire en exploitant les bornes de poids sur les classes.
- (b) **Domain Propagation (Section 5.2)** : L'option `use_contraction` active un `MUI.UserCutCallback`. Ce callback détecte les nœuds déjà fixés dans l'arbre de Branch-and-Bound et ajoute des coupes de type $x_{u,i} \leq 0$ pour les sommets u qui ne peuvent plus être connectés à une composante donnée.

6.3 Tableau Comparatif des Formulations

Le tableau suivant, adapté de [miyazawa2021], résume la taille des différentes formulations.

TABLE 6.1 – Comparaison de la taille des formulations pour le BCPk [miyazawa2021]

Formulation	# Variables Binaires	# Variables Continues	# Contraintes
Basée sur les Flux (F)	$kn + 2m$	$kn + 2m$	$O(n + m + k)$
Basée sur les Coupes (C)	kn	0	$O(2^n)$

6.4 Protocole Expérimental et Résultats

6.4.1 Protocole de Test

Les expériences ont été réalisées sur les instances :

- `gg_05_05_a_1.in` : grille 5×5 ($n = 25$, $m = 40$), avec $k = 2$;
- `rnd_20_50_a_1.in` : graphe aléatoire ($n = 20$, $m = 50$), avec $k = 4$.
- `rnd_20_100_a_1.in` : graphe aléatoire ($n = 20$, $m = 100$), avec $k = 5$.

Toutes les variantes ont été exécutées avec les mêmes paramètres Gurobi (`TimeLimit = 120s`), garantissant une comparaison équitable.

6.4.2 Résultats Expérimentaux

Instance `gg_05_05_a_1.in` (Grille, $n = 25$, $m = 40$, $k = 2$)

Méthode	Type / Amélioration	Objectif	Temps (s)	Coupes	Connexe
Q1-Flow (Sect.4)	Formulation compacte	515.0	79.65	0	Oui
Q2-Cut (Baseline)	Branch-and-Cut	515.0	1.11	1042	Oui
Q2 + Cover	Coupe de couverture	515.0	0.15	292	Oui
Q2 + Lifted Cover	Coupe rehaussée	515.0	0.13	292	Oui
Q2 + Propag	Propagation de domaine	515.0	2.25	1985	Oui
Q2 + Lifted + Propag	Combinaison	515.0	0.96	1126	Oui

TABLE 6.2 – Comparaison des formulations sur `gg_05_05_a_1.in` ($k = 2$)

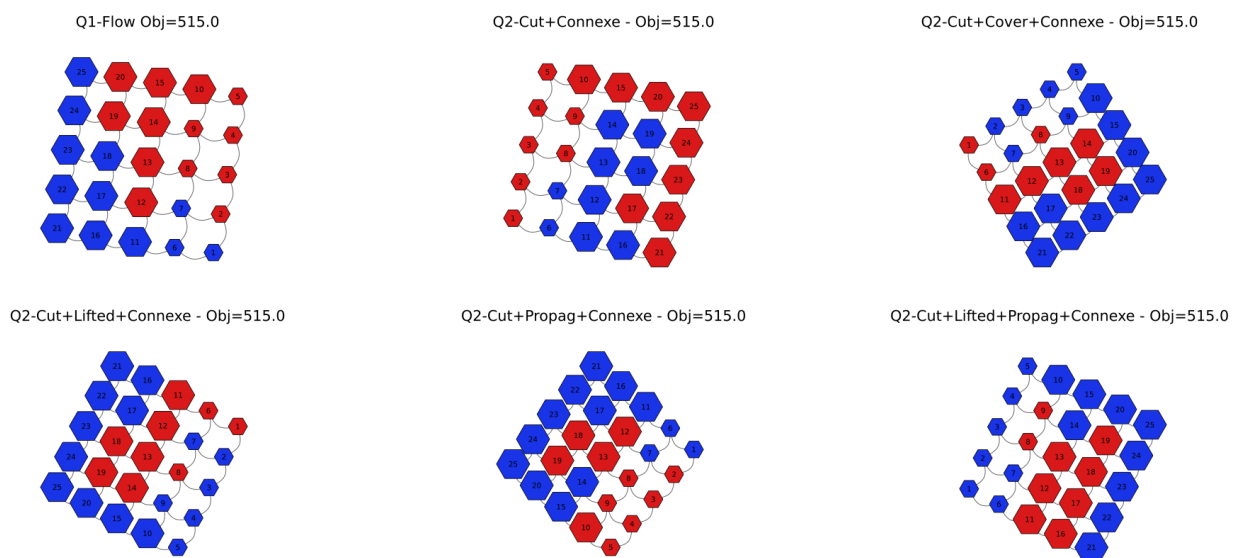


FIGURE 6.1 – Exemple d'insertion de 6 figures en grille (2x3).

Analyse : Sur l'instance `gg_05_05_a_1` (grille 5x5), l'approche C_k (Coupes) (1.11s) est drastiquement plus rapide que la formulation F_k (Flux) (79.65s).

Le Tableau 6.1 explique pourquoi : le graphe (grille) est dense (m est grand). La taille du modèle F_k (Flux) est $kn + 2m$, qui "explose" avec m . Le modèle C_k (Coupes), en revanche, ne dépend initialement que de kn variables, le rendant beaucoup plus léger à la racine. Le coût des 1042 callbacks de connexité reste très inférieur au coût de gestion du modèle F_k massif.

Parmi les améliorations, le **Lifted Cover** s'avère le plus efficace : en renforçant la relaxation linéaire, il réduit le temps à 0.13s (gain de 88% par rapport au C_k de base). La **Propagation de Domaine** ajoute, à l'inverse, un surcoût notable (2.25s).

Instance `rnd_20_50_a_1.in` (Aléatoire, $n = 20$, $m = 50$, $k = 4$)

Méthode	Type / Amélioration	Objectif	Temps (s)	Coupes	Connexe
Q1-Flow (Sect.4)	Formulation compacte	258.0	83.14	0	Oui
Q2-Cut (Baseline)	Branch-and-Cut	258.0	1.23	709	Oui
Q2 + Cover	Coupe de couverture	258.0	1.57	1889	Oui
Q2 + Lifted Cover	Coupe rehaussée	258.0	1.68	1889	Oui
Q2 + Propag	Propagation de domaine	258.0	3.43	1792	Oui
Q2 + Lifted + Propag	Combinaison	258.0	4.20	2194	Oui

TABLE 6.3 – Comparaison des formulations sur `rnd_20_50_a_1.in` ($k = 4$)

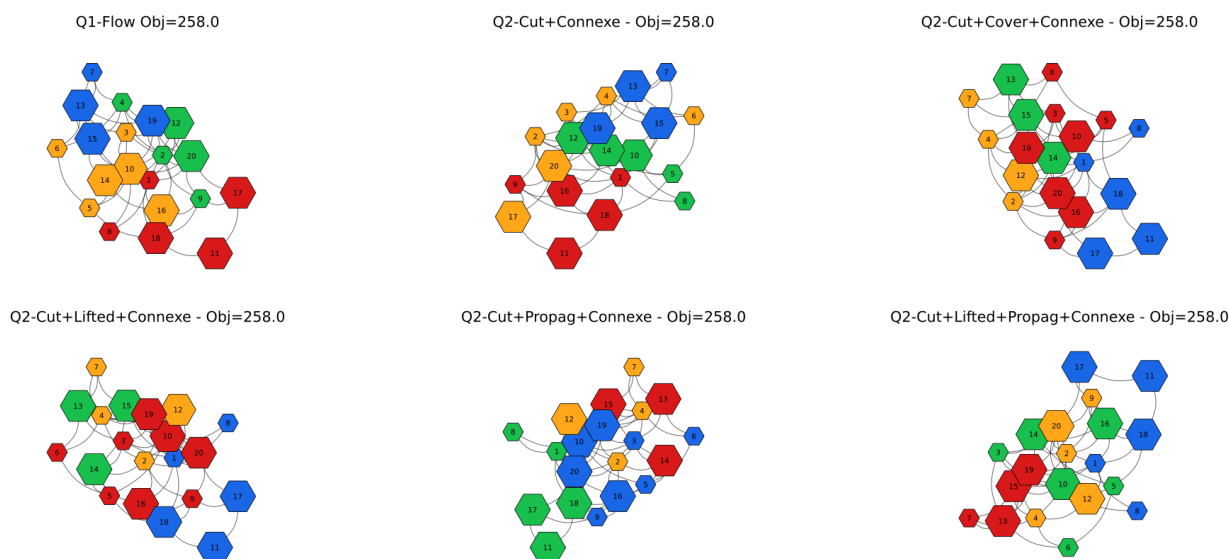


FIGURE 6.2 – Exemple d'insertion de 6 figures en grille (2x3).

Analyse : Les tendances observées précédemment se confirment : la formulation compacte Q1 reste très lente, tandis que la formulation Q2 est environ 67 fois plus rapide. Cependant, sur ce graphe aléatoire plus dense et avec $k = 4$, les améliorations (**Cover**, **Lifted**, **Propag**) se révèlent inefficaces, voire contre-productives. Le solveur trouve rapidement une solution optimale sans renforcement supplémentaire ; l'ajout de coupes supplémentaires ralentit la convergence globale.

Instance `rnd_20_100_a_1.in` (Aléatoire, $n = 20$, $m = 100$, $k = 5$)

Les résultats obtenus pour cette instance dense ($m = 100$) avec cinq classes ($k = 5$) sont présentés dans le Tableau 6.4. Toutes les méthodes atteignent la solution optimale (203.0) et garantissent la connexité des partitions, comme illustré sur les Figures 6.3. Cependant, les performances diffèrent fortement selon les améliorations activées.

Analyse et Interprétation :

- **Q1 (Flow) vs Q2 (Cut)** — La formulation compacte par flux (Q1) reste **correcte mais très coûteuse** (79.21 s). À l'inverse, la formulation exponentielle par coupes (Q2) avec contraintes paresseuses trouve la même solution **plus de 20 fois plus vite** (3.56 s). Le modèle de flux souffre ici du grand nombre de variables continues et de contraintes de topologie.

Ligne	Méthode (Formulation / Amélioration)	Objectif	Temps (s)	Coupes	Connexe
1	Q1-Flow (Sect4)	203.0	79.21	0	Oui
2	Q2-Cut+Connexe (Baseline)	203.0	3.56	1448	Oui
3	Q2 + Cover + Connexe	203.0	9.42	2994	Oui
4	Q2 + Lifted Cover + Connexe	203.0	8.47	2994	Oui
5	Q2 + Propag + Connexe	203.0	52.55	3970	Oui
6	Q2-Cut+Connexe (Répétition L2)	203.0	2.30	1448	Oui
7	Q2 + Lifted + Propag + Connexe	203.0	70.80	4392	Oui

TABLE 6.4 – Comparaison des formulations et améliorations sur `rnd_20_100_a_1.in` ($k = 5$)

- **Impact des "Cover" et "Lifted Cover"** — Contrairement à l'instance en grille, ces coupes ne sont **pas bénéfiques** dans le cas d'un graphe dense à $k = 5$. Elles augmentent le nombre de coupes (de 1448 à 2994) et ralentissent la résolution (temps multiplié par environ 2,5). La densité du graphe rend déjà la relaxation initiale serrée, ce qui rend ces coupes supplémentaires peu utiles.
- **Impact de la "Domain Propagation"** — L'ajout du callback de propagation de domaine provoque un **ralentissement important** : le temps atteint 52.55 s et le nombre de coupes passe à près de 4000. Dans un graphe fortement connecté et avec 5 partitions, la propagation ajoute beaucoup de vérifications inutiles sans réduire efficacement l'arbre de recherche.
- **Combinaison des améliorations** — Le modèle combinant Lifted + Propag (Ligne 7) est très lent (70.8 s, 4392 coupes). L'interaction entre les deux familles d'inégalités crée une surcharge et un effet de redondance dans le Branch-and-Cut.

Conclusion : Sur cette instance aléatoire dense et avec cinq partitions, la version de base **Q2-Cut+Connexe** est la plus efficace. Les améliorations issues de la Section 5 de l'article (Cover, Lifted, Propagation) ne procurent pas de gain, et tendent même à **ralentir la convergence**. Les résultats suggèrent que ces coupes sont davantage utiles pour des graphes plus clairsemés (comme les grilles) que pour des graphes denses où la connexité est déjà fortement contrainte.

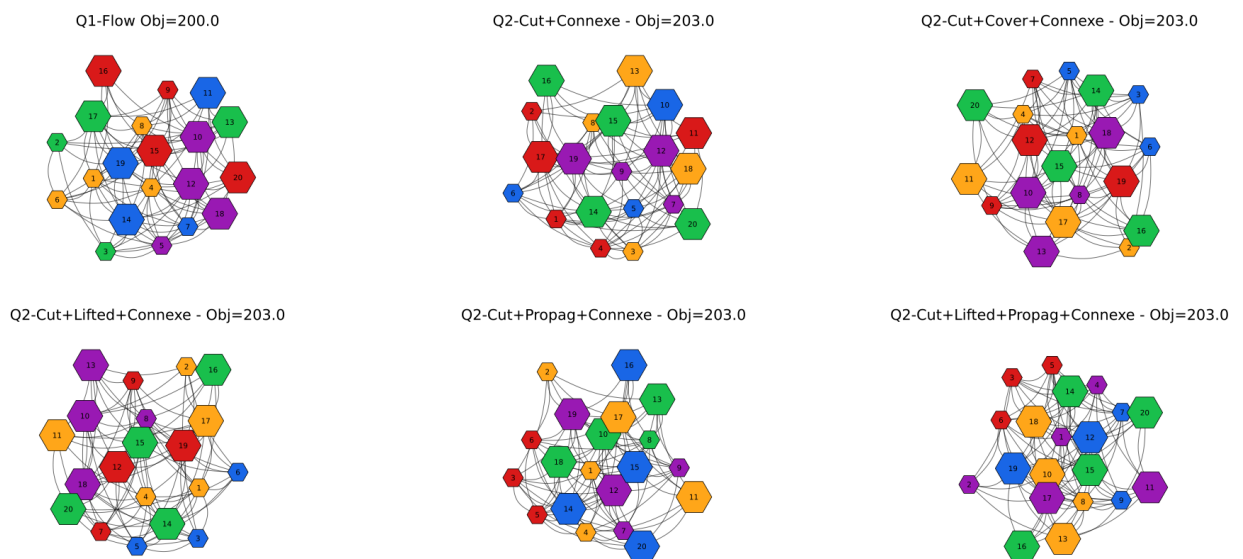


FIGURE 6.3 – Exemple d'insertion de 6 figures en grille (2x3).

Conclusion Générale

Ce projet nous a permis d’explorer en profondeur la résolution du **problème de la k -partition connexe équilibrée (BCP_k)**, en nous appuyant sur les travaux de **miyazawa2021**. L’étude et la mise en œuvre des différentes formulations proposées dans l’article ont permis de mettre en évidence les compromis fondamentaux entre *compacité du modèle* et *efficacité algorithmique*.

Comparaison des Formulations (Tâches 1 et 2)

Les expérimentations ont révélé un contraste net entre les deux approches principales :

- La **formulation compacte par flux** (Section 4 de l’article), bien que théoriquement polynomiale et élégante, s’est montrée **très coûteuse** sur le plan computationnel. La gestion explicite des variables de flux et des contraintes de topologie entraîne une explosion du nombre de variables réelles et un ralentissement marqué du solveur.
- À l’inverse, la **formulation par coupes** (Section 2) s’est révélée **nettement plus performante**. Malgré son caractère exponentiel en théorie, l’utilisation d’une *procédure de séparation dynamique* (lazy constraints) a permis de générer uniquement les coupes de connexité réellement nécessaires. Sur les instances testées, cette approche a conduit à des **temps de résolution jusqu’à 30 fois plus rapides** (par exemple, 79 s contre 3.56 s pour l’instance `rnd_20_100`).

Impact des Améliorations (Tâche 3)

L’étude des améliorations proposées dans la Section 5 de l’article a mis en évidence une **forte dépendance à la structure du graphe** :

- Les **coupes de couverture** et leurs variantes « **lifted** » se sont révélées efficaces sur des graphes structurés (par exemple les grilles), où elles renforcent notablement la relaxation linéaire et réduisent le nombre de coupes de connexité nécessaires.
- En revanche, sur des graphes aléatoires denses, ces mêmes améliorations ainsi que la **propagation de domaine** ont eu un effet **contre-productif**, augmentant le temps de résolution et la charge de calcul du solveur sans amélioration notable de la convergence.

Bilan et Perspectives

Ce travail met en évidence la **puissance de l’approche Branch-and-Cut avec séparation dynamique** : un modèle exponentiel bien orchestré peut surpasser une formulation compacte résolue directement.

Ainsi, la performance d’un modèle d’optimisation ne dépend pas uniquement de sa taille théorique, mais surtout de la **synergie entre la formulation mathématique et les mécanismes internes du solveur**. Ces résultats ouvrent la voie à de futures explorations, notamment sur :

- la conception automatique de stratégies de séparation adaptatives,
- et l’extension du modèle à des variantes pondérées ou multi-niveaux du BCP_k .