République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Akli Mohand Oulhadj – Bouira

Faculté des Sciences Exactes

Département de Mathématiques

Mémoire de Master

Spécialité : Recherche Opérationnelle

Thème :

Optimisation de l'affectation des enseignants aux modules par couplage dans les graphes bipartis

Réalisé par :

Khabou Fatma

Khelif Zineb

Devant le jury composé de :

M. Hamid	Karim	Président	MCB UAMO Bouira
Mme. Boudane	Khadidja	Examinatrice	MAA UAMO Bouira
Mme. Louadj	Kahina	Examinatrice	MCA UAMO Bouira
Mme. Imine	Ouiza	Encadrante	MAA UAMO Bouira

Année Universitaire 2025 / 2026

Résumé

Ce mémoire s'inscrit dans le cadre de la recherche opérationnelle et porte sur la modélisation et la résolution du problème d'affectation des enseignants aux modules au sein du département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira. Le problème est formulé à l'aide de la théorie des graphes, et plus précisément des graphes bipartis pondérés, dans lesquels une fonction de score d'adéquation est définie à partir de la spécialité, du grade et des préférences pédagogiques de chaque enseignant.

Trois approches de résolution sont étudiées et comparées : l'algorithme de Hopcroft–Karp pour le couplage maximum dans le cas non pondéré, l'algorithme Hongrois (Kuhn–Munkres) pour le cas pondéré, ainsi qu'une formulation générale en programmation linéaire en nombres entiers (PLNE) résolue à l'aide du solveur GLPK. L'ensemble des contraintes pédagogiques et administratives—habilitations, charges horaires, grades, préférences—est intégré au modèle.

L'implémentation des trois méthodes a été réalisée en Python, et une interface graphique conviviale a été développée afin de faciliter leur utilisation. Les expérimentations menées sur des données réelles montrent que la formulation PLNE fournit la solution la plus complète, en respectant l'ensemble des contraintes du problème, tandis que les algorithmes de Hopcroft–Karp et Hongrois constituent des outils efficaces pour des cas simplifiés.

Mots-clés : Recherche opérationnelle, théorie des graphes, graphes bipartis, couplage maximum, problème d'affectation, algorithme de Hopcroft–Karp, algorithme Hongrois, programmation linéaire en nombres entiers, GLPK, optimisation pédagogique.

Abstract

This thesis falls within the field of operations research and addresses the modelling and resolution of the teacher-to-course assignment problem within the Department of Mathematics at Akli Mohand Oulhadj University of Bouira. The problem is formulated using graph theory, and more specifically weighted bipartite graphs, in which a suitability score function is defined based on the specialty, the academic rank and the pedagogical preferences of each teacher.

Three resolution approaches are studied and compared : the Hopcroft–Karp algorithm for maximum cardinality matching in the unweighted case, the Hungarian algorithm (Kuhn–Munkres) for the weighted case, and a general formulation as an integer linear program (ILP) solved using the GLPK solver. The full set of pedagogical and administrative constraints—qualifications, teaching load limits, ranks, preferences—is incorporated into the model.

The three methods have been implemented in Python, and a user-friendly graphical interface has been developed to facilitate their use. Experiments carried out on real data show that the ILP formulation provides the most comprehensive solution by respecting all the constraints of the problem, while the Hopcroft–Karp and Hungarian algorithms remain efficient tools for simplified instances.

Keywords : Operations research, graph theory, bipartite graphs, maximum matching, assignment problem, Hopcroft–Karp algorithm, Hungarian algorithm, integer linear programming, GLPK, educational optimization.

Remerciements

Nous tenons tout d'abord à remercier Dieu Tout-Puissant de nous avoir accordé la santé, la force, la patience et la persévérance nécessaires pour mener ce travail à son terme.

Nous adressons nos sincères remerciements à notre encadrante, **Mme. IMINE Ouiza**, pour son accompagnement, sa disponibilité, ses précieux conseils ainsi que la confiance qu'elle nous a accordée tout au long de la réalisation de ce mémoire.

Nous exprimons également notre profonde gratitude à **Mme. LOUADJ Kahina** pour son soutien, sa bienveillance, ses encouragements et son aide précieuse, qui ont constitué une véritable source de motivation durant ce travail.

Nous remercions chaleureusement **Monsieur le président HAMID Karim**, ainsi que les membres du jury, **Mme .LOUADJ Kahina** et **Mme. BOUDANE Khadidja**, pour le temps qu'ils ont consacré à l'évaluation de ce mémoire, ainsi que pour leurs remarques pertinentes, leurs observations constructives et leurs précieux conseils, qui ont grandement contribué à l'amélioration et à l'enrichissement de ce travail.

Nous remercions également l'ensemble des enseignants du département de mathématiques de **l'Université Akli Mohand Oulhadj de Bouira** pour la qualité de la formation et des connaissances qu'ils nous ont transmises tout au long de notre parcours universitaire.

Enfin, nous adressons nos plus sincères remerciements à nos familles et à nos proches pour leur soutien, leurs encouragements et leur présence constante tout au long de ce parcours.

Dédicace

Je dédie ce modeste travail :

À mes très chers parents, *pour leur amour, leurs sacrifices et leur soutien durant tout mon parcours.*

À mes frères et sœurs, *pour leurs encouragements et leur présence à mes côtés.*

À mes amis et à toutes les personnes qui m'ont soutenue de près ou de loin.

À moi-même, *pour ma patience, ma persévérance et tous les efforts fournis afin d'arriver à cette étape.*

Dédicace

Avec une profonde gratitude, je dédie ce modeste travail :

À moi-même et à ma chère binôme,

Pour notre collaboration, notre patience et notre persévérance tout au long de ce parcours universitaire.

Je nous souhaite un avenir rempli de réussite et de bonheur.

À mes très chers parents,

Pour leur amour, leurs sacrifices, leur soutien et leurs encouragements constants.

Vous avez toujours été ma source de force et de motivation.

À ma chère sœur, SARA,

Pour sa présence, sa gentillesse et son soutien précieux.

À mon cher fiancé,

Pour sa compréhension, son soutien moral et ses encouragements tout au long de ce parcours.

À mes chers amis,

Pour leur sincerity, leur soutien et les beaux moments partagés.

Table des matières

Table des figures

Liste des tableaux

Liste des symboles

Symbole	**Signification**
$G = (X, E)$	Graphe non orienté de sommets X et d'arêtes E
$G = (X, U)$	Graphe orienté de sommets X et d'arcs U
$G = (X, Y, E)$	Graphe biparti de classes X et Y et d'arêtes E
$d_G(x)$	Degré du sommet x dans le graphe G
$\|E\|$	Nombre d'arêtes du graphe
M	Couplage dans un graphe biparti
$M_{\max}$	Couplage maximum
$\nu(G)$	Cardinal d'un couplage maximum de G
$\tau(G)$	Cardinal d'une couverture par sommets minimale de G
$\mathcal{E}$	Ensemble des enseignants
$\mathcal{M}$	Ensemble des modules
A	Ensemble des arêtes du graphe d'habilitation
n	Nombre d'enseignants
k	Nombre de modules
x_{ij}	Variable de décision binaire (1 si l'enseignant i est affecté au module j)
c_{ij}	Coût ou score d'adéquation entre i et j
C_{ij}	Score d'adéquation pondéré
h_{ij}	Indicateur d'habilitation (0 ou 1)
VH_j	Volume horaire du module j
$CH_i^{\max}$	Charge horaire maximale de l'enseignant i
G_i	Grade de l'enseignant i

Symbole	**Signification**
$G_j^{\min}$	Grade minimum requis pour le module j
s_{spec}	Score de spécialité
s_{grade}	Score de grade
s_{pref}	Score de préférence
α, β, γ	Pondérations ($\alpha + \beta + \gamma = 1$)
Z	Fonction objectif

TABLE 1 – Liste des principaux symboles utilisés dans ce mémoire

Notations et abréviations

Abréviation	**Signification**
RO	Recherche Opérationnelle
PLNE	Programmation Linéaire en Nombres Entiers
PL	Programmation Linéaire
GLPK	GNU Linear Programming Kit (solveur)
BFS	Breadth-First Search (parcours en largeur)
DFS	Depth-First Search (parcours en profondeur)
CSV	Comma-Separated Values
CM	Cours Magistral
TD	Travaux Dirigés
TP	Travaux Pratiques
EDP	Équations aux Dérivées Partielles
L1, L2, L3	Licence 1re, 2e, 3e année
S1, ..., S5	Semestres 1 à 5
MAA	Maître Assistant classe A
MCA	Maître de Conférences classe A
MCB	Maître de Conférences classe B

Table 2 – Liste des notations et abréviations utilisées

Introduction générale

La recherche opérationnelle s'est progressivement imposée comme un outil essentiel d'aide à la décision dans des contextes complexes faisant intervenir de nombreuses contraintes. Elle repose sur l'élaboration de modèles mathématiques et la mise en œuvre d'algorithmes performants, afin de proposer des solutions optimisées à des problèmes issus du monde réel.

Parmi les approches les plus utilisées dans ce domaine, la théorie des graphes offre un cadre particulièrement adapté à la modélisation de relations entre différents éléments. En représentant un système sous forme de sommets et de liens, elle permet de mieux comprendre sa structure et de concevoir des méthodes efficaces pour la résolution de nombreux problèmes d'optimisation.

Dans ce cadre, le problème du couplage occupe une place importante. Il consiste à établir des correspondances entre deux ensembles dans le respect de certaines contraintes, avec pour objectif l'optimisation d'un critère donné. Ce type de problème apparaît naturellement dans de nombreuses situations d'affectation, en particulier lorsqu'il s'agit d'associer des ressources à des tâches.

L'affectation des enseignants aux modules d'enseignement constitue un exemple concret de cette problématique. Dans les établissements universitaires, cette tâche est généralement réalisée en tenant compte de plusieurs facteurs : spécialités des enseignants, disponibilités, charges horaires et préférences individuelles. Cependant, une gestion exclusivement manuelle peut conduire à des résultats perfectibles, notamment en termes d'équilibre des charges ou d'adéquation entre compétences et modules.

Dans cette optique, le recours à une modélisation par graphes bipartis apparaît pertinent : les enseignants et les modules y sont représentés comme deux ensembles disjoints, et les arêtes traduisent les habilitations possibles. L'utilisation d'algorithmes de couplage adaptés offre alors la possibilité de déterminer une affectation à la fois cohérente et optimisée.

La question centrale abordée dans ce mémoire est ainsi la suivante : comment exploiter les outils de la théorie des graphes et de l'optimisation combinatoire pour améliorer le processus d'affectation des enseignants aux modules, tout en respectant les contraintes imposées par le contexte universitaire ?

Pour répondre à cette problématique, plusieurs objectifs ont été fixés. Il s'agit, dans

un premier temps, de présenter les notions fondamentales relatives aux graphes bipartis et au problème du couplage. Nous nous intéressons ensuite aux principaux algorithmes utilisés pour résoudre ce type de problème—l'algorithme de Hopcroft–Karp et l'algorithme Hongrois—ainsi qu'à la formulation en programmation linéaire en nombres entiers (PLNE), en mettant en évidence leurs caractéristiques respectives. Une modélisation du problème d'affectation est ensuite proposée, suivie de son implémentation en langage Python à l'aide du solveur GLPK. Enfin, une analyse des résultats obtenus permet d'évaluer l'efficacité des approches adoptées.

Ce mémoire est organisé en quatre chapitres.

— Le **premier chapitre** introduit les concepts de base de la théorie des graphes, en se concentrant sur les graphes bipartis, les couplages et leurs propriétés fondamentales.

— Le **deuxième chapitre** est consacré à l'étude des algorithmes de couplage—notamment l'algorithme de Hopcroft–Karp et l'algorithme Hongrois—ainsi qu'à la formulation du problème en programmation linéaire en nombres entiers (PLNE).

— Le **troisième chapitre** présente la modélisation détaillée du problème d'affectation des enseignants aux modules dans le contexte du département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira.

— Le **quatrième chapitre** décrit l'implémentation réalisée en Python, accompagnée d'une interface graphique, et présente les résultats expérimentaux obtenus sur des données réelles, ainsi qu'une analyse comparative des trois approches.

Une conclusion générale vient clore ce travail en synthétisant les principaux apports et en proposant quelques perspectives d'amélioration.

Chapitre 1

Concepts fondamentaux de la théorie des graphes

La théorie des graphes, née avec Euler et son célèbre problème des ponts de Königsberg en 1736, constitue l'un des piliers des mathématiques discrètes. Elle offre un cadre puissant pour modéliser de nombreux problèmes, parmi lesquels celui de l'affectation des enseignants aux modules, qui peut être formalisé à l'aide des graphes bipartis et des couplages maximaux.

Ce chapitre rappelle les notions mathématiques essentielles à la bonne compréhension des développements ultérieurs de ce mémoire. Pour un exposé plus approfondi, le lecteur pourra se reporter aux ouvrages spécialisés cités en bibliographie.

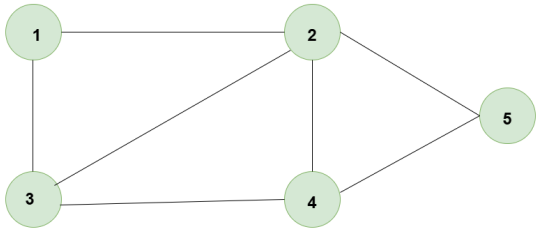
1.1 Définitions et notations de base

Définition 1.1 (Graphe)**.** *Un graphe est une représentation géométrique constituée d'un ensemble de points (appelés sommets ou nœuds) et d'un ensemble de lignes ou de flèches (appelées arêtes ou arcs) reliant ces points entre eux. Chaque arête possède pour extrémités deux sommets, qui peuvent éventuellement être confondus.*

Définition 1.2 (Graphe non orienté)**.** *Un graphe non orienté est défini par le couple $G = (X, E)$, où X est l'ensemble (fini) des sommets du graphe et E l'ensemble de ses arêtes.*

Si x et y sont deux sommets en relation, cette relation est décrite par l'arête $e = \{x, y\}$. Le sens de la relation n'est pas pris en compte.

Exemple :



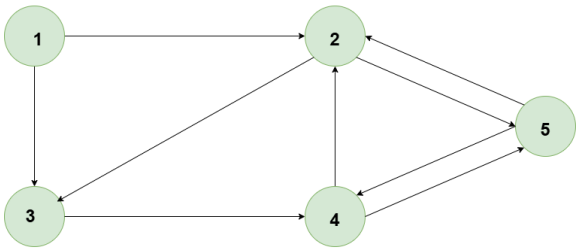
G=(X,E)
X={1,2,3,4,5}
E={{1,2},{1,3},{2,3},{2,4},{2,5},{3,4},{4,5}}

FIGURE 1.1 – Graphe non orienté

Définition 1.3 (Graphe orienté)**.** *Un graphe orienté est un couple $G = (X, U)$, où X est l'ensemble (fini) des sommets et U l'ensemble (fini) de ses arcs.*

Un arc $u = (x, y) \in U$, avec $x, y \in X$, représente une relation orientée du sommet x vers le sommet y.

Exemple :



G=(X,U)
X={1,2,3,4,5}
U={{1,2},{1,3},{2,3},{3,4},{4,2},{2,5},{5,2},{4,5},{5,4}}

FIGURE 1.2 – Graphe orienté

Définition 1.4 (Graphe pondéré)**.** *Un graphe pondéré est un triplet $G = (X, E, w)$ où (X, E) est un graphe (orienté ou non) et*

$$w : E \longrightarrow \mathbb{R}$$

est une fonction de pondération qui associe à chaque arête $e \in E$ un poids réel $w(e)$ représentant une grandeur telle qu'une distance, un coût ou un temps.

Exemple :

Les valeurs 10,12,14,15,20,11 représentent les poids associés aux arêtes du graphe .

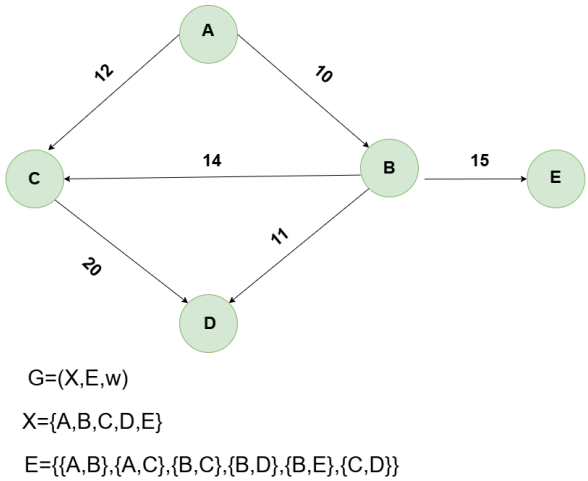


FIGURE 1.3 – Graphe pondéré

Définition 1.5 (Degré d'un sommet)**.** *On appelle degré d'un sommet x d'un graphe non orienté G le nombre d'arêtes de G incidentes à x, c'est-à-dire admettant x comme extrémité. Chaque boucle (arête dont les deux extrémités coïncident) compte deux fois. On note $d(x)$ ou $d_G(x)$ ce nombre entier.*

Un sommet est dit isolé si son degré est nul.

Théorème 1.6 (Lemme des poignées de main)**.** *La somme des degrés des sommets d'un graphe non orienté $G = (X, E)$ est égale à deux fois le nombre de ses arêtes :*

$$\sum_{x \in X} d_G(x) = 2\,|E|.$$

Définition 1.7 (Chaîne et chemin)**.** *Soit $G = (X, E)$ un graphe non orienté. Une chaîne reliant deux sommets x_0 et x_k est une suite de sommets $(x_0, x_1, x_2, \ldots, x_k)$ telle que, pour tout $i \in \{0, 1, \ldots, k-1\}$, $\{x_i, x_{i+1}\} \in E$.*

Dans le cas d'un graphe orienté $G = (X, U)$, on parle de chemin de x_0 à x_k : il s'agit d'une suite $(x_0, x_1, \ldots, x_k)$ telle que, pour tout i, $(x_i, x_{i+1}) \in U$.

Exemple :

Dans le graphe ci-dessous, la suite de sommets (A, D, C, E) est une chaîne joignant A à E.

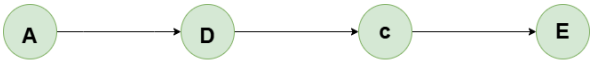


FIGURE 1.4 – Exemple de chaîne

Définition 1.8 (Cycle). *Un cycle est une chaîne simple $(x_0, x_1, \ldots, x_k)$ dont les deux extrémités coïncident, c'est-à-dire telle que $x_0 = x_k$, et dont les arêtes sont deux à deux distinctes.*

Exemple :

Dans le graphe ci-dessous, la suite de sommets (A, B, C, D, A) est un cycle.

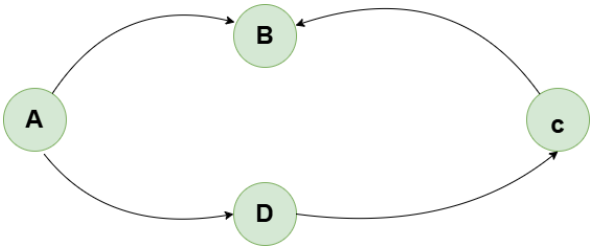


FIGURE 1.5 – Exemple de cycle

Définition 1.9 (Connexité). *Soit $G = (X, E)$ un graphe non orienté. Deux sommets x et y sont dits connectés s'il existe une chaîne reliant x à y, ou si $x = y$. Le graphe G est dit connexe si tous ses sommets sont deux à deux connectés.*

Exemple :

Il existe une chaîne $C = (x_1, x_3, x_2)$ entre x_1 et x_2 : ces deux sommets sont donc connectés. En revanche, il n'existe aucune chaîne entre x_1 et x_5 : ces deux sommets ne sont pas connectés.

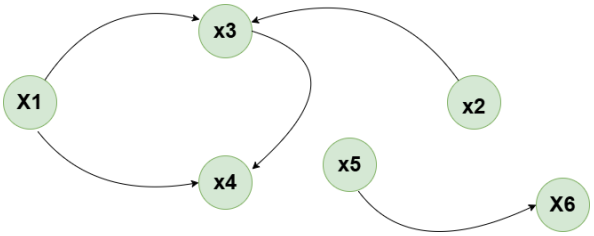


FIGURE 1.6 – Illustration de la connexité

1.2 Graphes bipartis

Définition 1.10 (Graphe biparti). *Un graphe G est dit biparti si son ensemble de sommets peut être partitionné en deux sous-ensembles disjoints, appelés classes, tels que toute arête possède une extrémité dans chacune des deux classes.*

On note un graphe biparti $G = (X, Y, E)$, où X et Y sont les deux classes de sommets (avec $X \cap Y = \emptyset$) et $E \subseteq X \times Y$ l'ensemble des arêtes.

Les graphes bipartis occupent une place importante en théorie des graphes, notamment dans diverses applications telles que les problèmes de couplage. Ils constituent un cadre naturel pour modéliser des relations entre deux types distincts d'entités. Dans le contexte du présent mémoire, l'ensemble X représentera les enseignants et l'ensemble Y les modules. Par ailleurs, les graphes bipartis présentent l'avantage d'admettre une caractérisation simple en termes de cycles.

Exemple :

On considère le graphe biparti de la figure 1.7, dont les classes sont $X = \{e_1, e_2, e_3, e_4\}$ et $Y = \{m_1, m_2, m_3, m_4\}$.

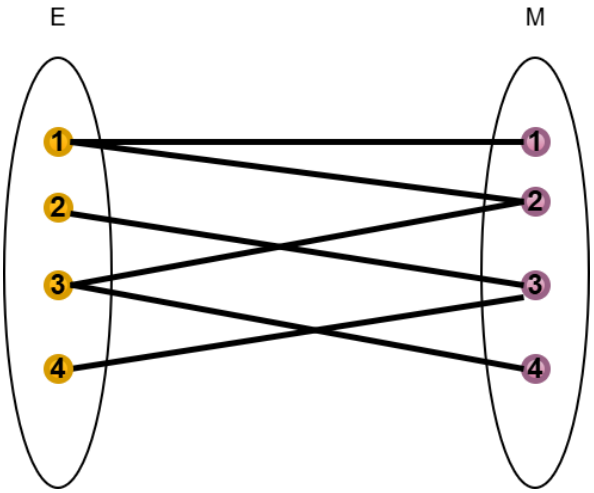


FIGURE 1.7 – Exemple de graphe biparti

Théorème 1.11 (Caractérisation des graphes bipartis)**.** *Un graphe est biparti si et seulement s'il ne contient aucun cycle de longueur impaire.*

1.2.1 Matrice d'adjacence d'un graphe biparti

Soit $G = (X, Y, E)$ un graphe biparti, avec $X = \{x_1, \ldots, x_n\}$ et $Y = \{y_1, \ldots, y_m\}$. On appelle matrice de bi-adjacence (ou simplement matrice d'adjacence) du graphe biparti G la matrice $A = (a_{ij})$ de taille $|X| \times |Y|$ définie par :

$$a_{ij} = \begin{cases} 1 & \text{si } \{x_i, y_j\} \in E, \\ 0 & \text{sinon.} \end{cases}$$

Dans le cas pondéré, on pose $a_{ij} = w(x_i, y_j)$ lorsque $\{x_i, y_j\} \in E$, et $a_{ij} = 0$ sinon. La valeur a_{ij} représente alors, dans notre contexte applicatif, le score d'adéquation entre l'enseignant x_i et le module y_j.

Exemple :

La matrice d'adjacence associée au graphe biparti de la figure 1.7 est définie par :

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

1.3 Couplage dans un graphe biparti

Définition 1.12 (Couplage)**.** *Soit $G = (X, Y, E)$ un graphe biparti. Un couplage M de G est un sous-ensemble d'arêtes $M \subseteq E$ tel que chaque sommet de G soit incident à au plus une arête de M.*

Exemple :

On considère le graphe biparti de la figure 1.8. Les arêtes $\{e_1, m_2\}$ et $\{e_3, m_4\}$, représentées en rouge, forment un couplage.

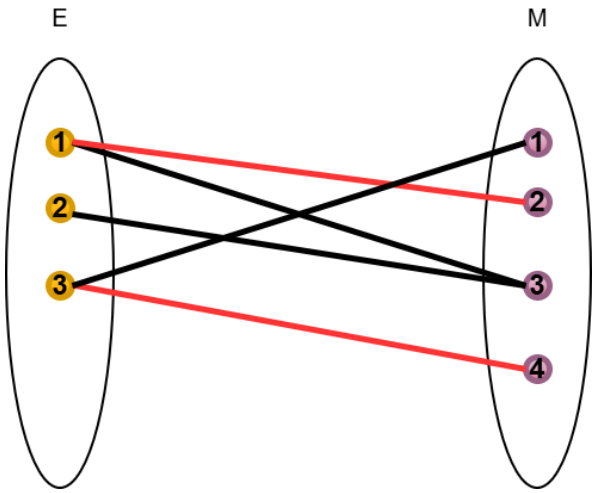


FIGURE 1.8 – Couplage d'un graphe biparti

Remarque. *Soit $G = (X, Y, E)$ un graphe biparti. Un couplage M de G peut être vu comme le graphe d'une **fonction injective** extraite de la correspondance représentée par G. Ainsi, il existe autant de couplages que de fonctions injectives extraites de cette correspondance.*

Étant donné un couplage $M \subseteq E$, il est parfois possible de construire un nouveau couplage M' en adjoignant à M une ou plusieurs arêtes de $E \setminus M$.

Définition 1.13 (Couplage maximal)**.** *Un couplage M d'un graphe biparti $G = (X, Y, E)$ est dit maximal (au sens de l'inclusion) s'il ne peut être agrandi par adjonction d'une arête de $E \setminus M$; autrement dit, pour toute arête $e \in E \setminus M$, l'ensemble $M \cup \{e\}$ n'est plus un couplage.*

Exemple :

Ce graphe illustre un couplage maximal M (représenté par les arêtes en noir) ainsi que les arêtes restantes appartenant à $E \setminus M$ (représentées en rouge).

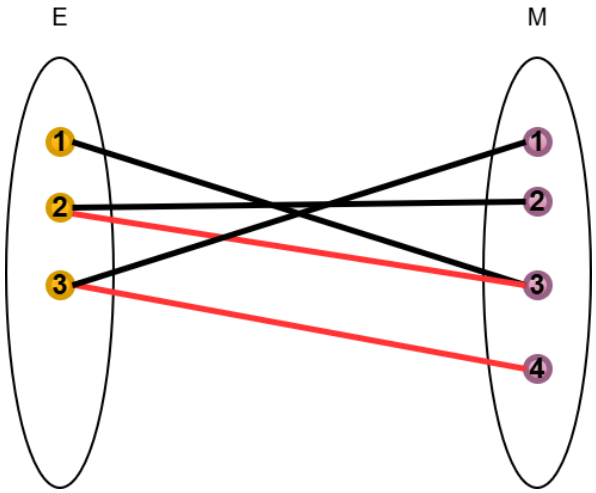


FIGURE 1.9 – Couplage maximal

Soit $G = (X, Y, E)$ un graphe biparti et soit $\mathcal{M}$ l'ensemble de tous ses couplages. L'ensemble $\mathcal{M}$ est fini, puisqu'il est inclus dans l'ensemble $\mathcal{P}(E)$ des parties de E, dont le cardinal vaut $2^{|E|}$.

Définition 1.14 (Couplage maximum)**.** *Soit $G = (X, Y, E)$ un graphe biparti et $\mathcal{M}$ l'ensemble de ses couplages. On appelle couplage maximum de G, et on note $M_{\max}$, tout couplage de cardinal maximal :*

$$|M_{\max}| = \max\Big\{\, |M| \;:\; M \in \mathcal{M} \,\Big\}.$$

Remarque. *Tout couplage maximum est maximal, mais la réciproque est fausse en général.*

Exemple :

Considérons le graphe biparti représenté dans la 1.10. Les arêtes en rouge représentent les arêtes du couplage maximum $M_{\max}$, où :

$$M_{\max} = \{(1, 3), (2, 2), (3, 4)\}$$

Ce couplage est de cardinal maximal $|M_{\max}| = 3$

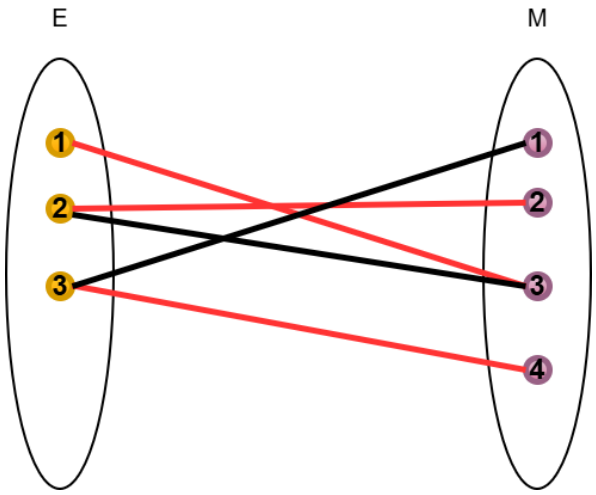


FIGURE 1.10 – Couplage maximum

Définition 1.15 (Couplage parfait)**.** *Soit $G = (X, Y, E)$ un graphe biparti tel que $|X| = |Y|$. Un couplage M de G est dit parfait si $|M| = |X|$, c'est-à-dire si tout sommet de G est incident à exactement une arête de M.*

Exemple :

Considérons le graphe biparti représenté dans la 1.11. Les arêtes en rouge définissent un couplage parfait M, où :

$$M = \{(1, 3), (2, 2), (3, 1), (4, 4)\}$$

avec $|M| = |X| = |Y| = 4$.

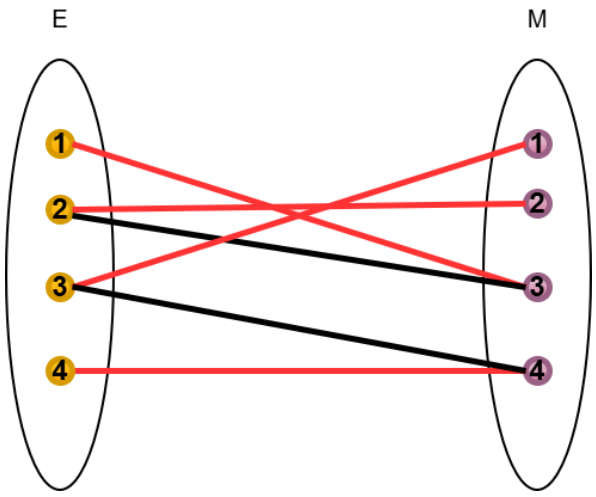


FIGURE 1.11 – Couplage parfait

Remarque. *Un couplage parfait peut être vu comme le graphe d'une application bijective extraite de la correspondance représentée par G. Décider de l'existence d'un couplage parfait dans un graphe biparti donné constitue une étape importante dans la résolution de nombreux problèmes faisant intervenir ce type de graphes : c'est le* ***problème du couplage biparti parfait****.*

1.3.1 Chaîne augmentante

Définition 1.16 (Chaîne alternante, chaîne augmentante)**.** *Soit $G = (X, Y, E)$ un graphe biparti et M un couplage de G.*

Une chaîne alternante (relativement à M) est une chaîne dont les arêtes appartiennent alternativement à M et à $E \setminus M$.

Une chaîne augmentante est une chaîne alternante dont les deux extrémités sont des sommets non saturés par M (c'est-à-dire des sommets non incidents à une arête de M).

FIGURE 1.12 – Chaîne alternante

FIGURE 1.13 – Chaîne augmentante

Lemme 1.17 (Lemme de Berge, 1957)**.** *Un couplage M d'un graphe G est maximum (c'est-à-dire de cardinal maximal) si et seulement s'il n'existe aucune chaîne augmentante relativement à M.*

Démonstration. ($\Rightarrow$) Supposons qu'il existe une chaîne augmentante P relativement à M. Par définition, P commence et se termine en des sommets non saturés, et alterne arêtes de M et arêtes de $E \setminus M$.

On définit alors $M' = M \triangle P$ (différence symétrique entre M et l'ensemble des arêtes de P) : on retire de M les arêtes de $P \cap M$ et on ajoute celles de $P \setminus M$. Le sous-ensemble M' ainsi obtenu est encore un couplage et vérifie $|M'| = |M| + 1$. Donc M n'est pas maximum.

($\Leftarrow$) Supposons que M ne soit pas maximum. Il existe alors un couplage M' tel que $|M'| > |M|$.

Considérons la différence symétrique $H = M \triangle M'$. Le sous-graphe induit par H ne contient que des sommets de degré au plus 2 (chacun étant incident à au plus une arête de M et une arête de M'). Ses composantes connexes sont donc des chaînes ou des cycles, dont les arêtes alternent entre M et M'. Les cycles alternants comportent

autant d'arêtes de M que de M'. Comme $|M'| > |M|$, il existe nécessairement au moins une chaîne dans H contenant strictement plus d'arêtes de M' que de M : ses extrémités sont alors non saturées par M, et cette chaîne est augmentante relativement à M. □

1.4 Théorème de König

Définition 1.18 (Couverture par sommets)**.** *Soit $G = (X, Y, E)$ un graphe biparti. Une couverture par sommets (ou transversal) est un sous-ensemble $C \subseteq X \cup Y$ tel que toute arête $e \in E$ soit incidente à au moins un sommet de C. Autrement dit, chaque arête du graphe est « touchée » par au moins un sommet de C.*

Théorème 1.19 (König, 1931)**.** *Soit $G = (X, Y, E)$ un graphe biparti. Alors*

$$\nu(G) = \tau(G),$$

où $\nu(G)$ désigne le cardinal d'un couplage maximum de G et $\tau(G)$ le cardinal d'une couverture par sommets minimale de G.

Ce résultat peut s'interpréter comme une égalité entre la valeur d'un flot maximum et la capacité d'une coupe minimale dans un certain réseau associé à G. Pour le voir, on construit un réseau R à partir du graphe biparti G de la manière suivante :

- on oriente chaque arête de G de X vers Y et on lui attribue une capacité infinie ;
- on introduit un sommet s jouant le rôle de source unique du réseau, relié à chaque sommet $x \in X$ par un arc (s, x) de capacité 1 ;
- on ajoute un sommet t jouant le rôle de puits unique, relié à chaque sommet $y \in Y$ par un arc (y, t) de capacité 1.

On observe alors les correspondances suivantes :

- tout flot entier dans R définit naturellement un couplage de G, et la valeur du flot est égale au cardinal de ce couplage (en considérant les arcs de flot égal à 1 entre X et Y) ;
- toute coupe finie de R correspond à une couverture par sommets de G, dont le cardinal coïncide avec la capacité de la coupe. Plus précisément, si la coupe est associée à un ensemble Z de sommets contenant s mais pas t, alors la couverture associée est $(X \setminus Z) \cup (Y \cap Z)$.

Ainsi, la valeur d'un flot maximum dans R coïncide avec la taille d'un couplage maximum de G, c'est-à-dire $\nu(G)$, tandis que la capacité d'une coupe minimum de R coïncide avec la taille d'une couverture par sommets minimum de G, c'est-à-dire $\tau(G)$. Le théorème de Ford–Fulkerson (égalité flot maximum – coupe minimum) implique alors directement l'égalité $\nu(G) = \tau(G)$, qui constitue précisément l'énoncé du théorème de König pour les graphes bipartis.

Conclusion

Ce chapitre a posé les fondements de la théorie des graphes en se focalisant sur les graphes bipartis et les couplages, qui constituent les piliers de notre problème d'affectation enseignants-modules. Nous avons rappelé les notions de base—graphes simples, degrés, chaînes, chemins et connexité—, puis défini les graphes bipartis par leur bipartition (équivalente à l'absence de cycle impair) et leur matrice de bi-adjacence, particulièrement adaptés à la modélisation de nos contraintes pédagogiques. Les notions de couplage maximal, maximum et parfait ont ensuite été explorées à travers les chaînes augmentantes, le lemme de Berge, le théorème de König et la réduction au flot maximum, fournissant ainsi des critères d'optimalité précis.

Ce socle théorique prépare le terrain pour le chapitre suivant, dédié aux algorithmes de couplage—notamment Hopcroft–Karp et l'algorithme hongrois—ainsi qu'à la formulation en programmation linéaire en nombres entiers, sur lesquels reposera notre démarche d'optimisation pédagogique.

Chapitre 2

Algorithmes de couplage et d'optimisation

Introduction

Avant d'aborder la modélisation détaillée du problème d'affectation des enseignants aux modules, il est nécessaire de présenter les outils théoriques et les méthodes algorithmiques permettant d'en assurer une résolution efficace.

Ce chapitre a ainsi pour objectif d'introduire le cadre algorithmique de référence, en s'appuyant sur l'étude des algorithmes de couplage dans les graphes bipartis, qui constituent le fondement des approches développées dans la suite du mémoire.

Deux cas principaux sont distingués selon la nature du problème :

— **Cas non pondéré** : l'objectif est de maximiser le nombre d'affectations possibles, sans prise en compte de leur qualité. Ce cas correspond au problème du couplage maximum en cardinalité dans un graphe biparti.

— **Cas pondéré** : des critères tels que les préférences et les compétences des enseignants sont intégrés, dans le but de maximiser la qualité globale de l'affectation. C'est le problème d'affectation classique.

Par ailleurs, une formulation en programmation linéaire en nombres entiers (PLNE) est introduite comme un cadre de modélisation général et flexible. Cette approche permet d'intégrer des contraintes plus complexes et d'obtenir des solutions mieux adaptées aux besoins réels.

L'objectif de ce chapitre est de poser les bases conceptuelles et algorithmiques nécessaires à la modélisation et à la résolution rigoureuse du problème étudié.

2.1 Algorithme de Hopcroft–Karp

2.1.1 Présentation de l'algorithme

L'algorithme de Hopcroft–Karp, proposé par John Hopcroft et Richard Karp en 1973, est une méthode efficace permettant de déterminer un couplage maximum dans un graphe biparti non pondéré.

Cet algorithme se distingue des approches classiques fondées sur la recherche successive de chaînes augmentantes par le fait qu'il traite, à chaque itération, un ensemble maximal de chaînes augmentantes disjointes de longueur minimale. Cette stratégie permet de réduire le nombre total d'itérations nécessaires et améliore significativement la performance globale.

2.1.2 Principe de l'algorithme

On considère un graphe biparti $G = (X, Y, E)$, où :

— X représente l'ensemble des sommets de la première classe (par exemple, les enseignants) ;

— Y représente l'ensemble des sommets de la seconde classe (par exemple, les modules) ;

— $E \subseteq X \times Y$ est l'ensemble des arêtes représentant les affectations possibles.

L'objectif est de déterminer un couplage maximum $M \subseteq E$.

L'algorithme repose sur l'alternance de deux phases principales :

— **Phase de recherche en largeur (BFS).** À partir des sommets libres de X, on construit un graphe en niveaux qui identifie les chaînes augmentantes de longueur minimale en alternant les arêtes appartenant à M et celles n'y appartenant pas.

— **Phase de recherche en profondeur (DFS).** Le graphe en niveaux est exploité pour rechercher un ensemble maximal de chaînes augmentantes disjointes. Chaque chaîne trouvée permet d'augmenter le couplage par symétrisation.

Ces deux phases sont répétées jusqu'à ce qu'il n'existe plus aucune chaîne augmentante. À ce moment, d'après le lemme de Berge, le couplage obtenu est maximum.

2.1.3 Pseudo-code de l'algorithme

Pour chaque sommet $x \in X$ (resp. $y \in Y$), on note $\text{Pair}_X[x]$ (resp. $\text{Pair}_Y[y]$) le sommet auquel il est apparié dans le couplage courant, ou NIL s'il est libre. On note également $\text{Adj}(x)$ la liste d'adjacence de x et $\text{dist}[x]$ son niveau dans le graphe construit par le BFS. Par convention, on définit aussi dist[NIL] : sa valeur, mise à jour par le BFS, sert de condition d'arrêt à la fois pour le BFS lui-même (qui s'arrête de propager

au-delà de ce niveau) et pour le DFS (qui ne suit que des arêtes augmentant strictement la distance).

Algorithm 1 Algorithme de Hopcroft–Karp

Require: Graphe biparti $G = (X, Y, E)$
Ensure: Couplage maximum M

```
Pair_X[x] ← NIL pour tout x ∈ X
Pair_Y[y] ← NIL pour tout y ∈ Y
|M| ← 0
while BFS() = vrai do
    for chaque x ∈ X tel que Pair_X[x] = NIL do
        if DFS(x) = vrai then
            |M| ← |M| + 1
        end if
    end for
end while
return |M| et le couplage codé par Pair_X, Pair_Y
```

Algorithm 2 Procédure BFS

```
1: Q ← file vide
2: for chaque x ∈ X do
3:     if Pair_X[x] = NIL then
4:         dist[x] ← 0 ; enfiler x dans Q
5:     else
6:         dist[x] ← +∞
7:     end if
8: end for
9: dist[NIL] ← +∞
10: while Q est non vide do
11:     défiler x de Q
12:     if dist[x] < dist[NIL] then
13:         for chaque y ∈ Adj(x) do
14:             if dist[Pair_Y[y]] = +∞ then
15:                 dist[Pair_Y[y]] ← dist[x] + 1
16:                 enfiler Pair_Y[y] dans Q
17:             end if
18:         end for
19:     end if
20: end while
21: return (dist[NIL] ≠ +∞)
```

Algorithm 3 Procédure DFS

```
1: if x ≠ NIL then
2:     for chaque y ∈ Adj(x) do
3:         if dist[Pair_Y[y]] = dist[x] + 1 then
4:             if DFS(Pair_Y[y]) = vrai then
5:                 Pair_Y[y] ← x
6:                 Pair_X[x] ← y
7:                 return vrai
8:             end if
9:         end if
10:    end for
11:    dist[x] ← +∞
12:    return faux
13: end if
14: return vrai
```

2.1.4 Complexité

La complexité temporelle de l'algorithme de Hopcroft–Karp est en

$$O\left(\sqrt{n} \cdot m\right),$$

où $V = X \cup Y$ désigne l'ensemble des sommets et E celui des arêtes du graphe.

Cette complexité est nettement meilleure que celle des méthodes classiques fondées sur la recherche itérative de chaînes augmentantes simples (algorithme de Ford–Fulkerson appliqué au couplage), dont la complexité est en $O(n \cdot m)$.

La complexité spatiale est en

$$O(n + m),$$

ce qui correspond au stockage du graphe, des appariements et des structures auxiliaires utilisées pendant l'exécution.

L'algorithme de Hopcroft–Karp constitue ainsi une méthode efficace pour le calcul d'un couplage maximum dans un graphe biparti non pondéré. Dans le cadre de ce travail, il est utilisé comme base pour analyser l'existence d'affectations possibles entre les enseignants et les modules avant d'introduire des critères d'optimisation pondérés.

2.2 Algorithme Hongrois

2.2.1 Présentation de l'algorithme

L'algorithme Hongrois est une méthode classique d'optimisation combinatoire, introduite par Harold Kuhn en 1955 et perfectionnée par la suite par James Munkres (d'où le nom alternatif d'algorithme de Kuhn–Munkres). Il est spécifiquement conçu pour résoudre le problème d'affectation dans un graphe biparti pondéré.

Soit une matrice de coût carrée $C = (c_{ij}) \in \mathbb{R}^{n \times n}$, où chaque élément c_{ij} représente le coût (ou, dualement, le gain) associé à l'affectation de l'élément i de la première classe à l'élément j de la seconde. L'objectif est de déterminer une bijection optimale entre les deux ensembles, c'est-à-dire une permutation σ de $\{1, \ldots, n\}$ minimisant (resp. maximisant) la quantité

$$\sum_{i=1}^{n} c_{i,\sigma(i)}.$$

Remarque. *Lorsque le problème est posé en termes de maximisation d'un score (par exemple, un score d'adéquation), il suffit de remplacer la matrice C par $C' = C_{\max} - C$ (où $C_{\max}$ est une borne supérieure des c_{ij}), ou plus simplement par $-C$, pour se ramener à une minimisation. C'est cette transformation qui sera utilisée dans le chapitre 4.*

2.2.2 Principe de l'algorithme

Le fonctionnement de l'algorithme Hongrois repose sur une transformation progressive de la matrice de coût. L'idée principale consiste à simplifier la matrice tout en conservant l'équivalence du problème initial, afin de faire apparaître explicitement des affectations optimales sous forme de zéros.

L'algorithme s'appuie sur les étapes suivantes.

1. Réduction de la matrice. Dans un premier temps, la matrice est normalisée :
- on soustrait à chaque ligne son minimum ;
- on soustrait ensuite à chaque colonne son minimum.

Cette transformation fait apparaître des zéros, qui correspondent à des affectations potentiellement optimales. La validité de cette opération repose sur le fait que soustraire une constante à une ligne ou à une colonne ne change pas la solution optimale du problème d'affectation.

2. Identification des affectations candidates. Les zéros obtenus sont exploités pour construire un ensemble d'affectations candidates, en respectant la contrainte d'indépendance : au plus un zéro sélectionné par ligne et par colonne.

3. Vérification de l'optimalité. Si l'on parvient à sélectionner n zéros indépendants, alors une affectation complète est obtenue et celle-ci est optimale. Dans le cas contraire, la matrice doit être ajustée.

4. Couverture minimale et ajustement. On recouvre l'ensemble des zéros par un nombre minimal de lignes et de colonnes. Soit k le plus petit élément non couvert. On effectue alors les opérations suivantes :
- soustraire k à tous les éléments non couverts ;
- ajouter k aux éléments situés à l'intersection de deux droites de couverture (ligne et colonne) ;
- laisser inchangés les autres éléments.

Cette opération crée de nouveaux zéros sans détruire les solutions admissibles déjà identifiées.

5. Itération. Les étapes précédentes sont répétées jusqu'à l'obtention d'une affectation complète à n zéros indépendants.

Ainsi, l'algorithme transforme progressivement la matrice initiale en une matrice équivalente dans laquelle la solution optimale apparaît sous forme d'un ensemble de zéros indépendants.

2.2.3 Pseudo-code de l'algorithme

Algorithm 4 Algorithme Hongrois (problème d'affectation)

Require: Matrice de coût $C \in \mathbb{R}^{n \times n}$

Ensure: Affectation optimale $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$

Étape 1 : Réduction des lignes
for $i = 1$ à n **do**
    $m_i \leftarrow \min_j C[i, j]$
    for $j = 1$ à n **do**
        $C[i, j] \leftarrow C[i, j] - m_i$
    end for
end for
Étape 2 : Réduction des colonnes
for $j = 1$ à n **do**
    $m_j \leftarrow \min_i C[i, j]$
    for $i = 1$ à n **do**
        $C[i, j] \leftarrow C[i, j] - m_j$
    end for
end for
while aucune affectation complète n'est trouvée **do**
    Couvrir tous les zéros par un nombre minimal ℓ de lignes et de colonnes
    if $\ell = n$ **then**
        break
    else
        $k \leftarrow$ minimum des éléments non couverts
        for chaque élément non couvert (i, j) **do**
            $C[i, j] \leftarrow C[i, j] - k$
        end for
        for chaque élément couvert deux fois (i, j) **do**
            $C[i, j] \leftarrow C[i, j] + k$
        end for
    end if
end while
Étape finale : sélectionner n zéros indépendants (un par ligne et par colonne)
return l'affectation optimale σ

2.2.4 Complexité de l'algorithme Hongrois

L'algorithme Hongrois présente une complexité temporelle en $O(n^3)$, où n désigne la dimension de la matrice de coût. Cette complexité est due aux différentes opérations de transformation de la matrice—en particulier les phases de réduction et d'ajustement—ainsi qu'à la recherche répétée d'une couverture minimale des zéros.

La complexité spatiale est, quant à elle, en $O(n^2)$: elle correspond au stockage de la matrice de coût et des structures auxiliaires nécessaires à l'exécution.

L'algorithme Hongrois constitue ainsi une approche efficace et largement utilisée pour résoudre le problème d'affectation optimale. Grâce à sa complexité polynomiale, il offre un bon compromis entre précision et performance, ce qui le rend particulièrement adapté à des applications pratiques de taille moyenne, comme celle étudiée dans ce mémoire.

2.3 Formulation en programmation linéaire en nombres entiers (PLNE)

2.3.1 Définition et principes généraux

La programmation linéaire en nombres entiers (PLNE) est un cadre de modélisation mathématique largement utilisé pour représenter et résoudre des problèmes d'optimisation combinatoire. Elle se distingue de la programmation linéaire classique par l'introduction de contraintes d'intégrité imposant aux variables de décision de prendre des valeurs entières.

Cette caractéristique permet de modéliser des situations réelles où les décisions sont de nature discrète, en particulier lorsque les choix sont binaires (variables à valeurs dans $\{0, 1\}$) ou entiers.

Un modèle de PLNE repose sur trois composantes fondamentales :

- un ensemble de **variables de décision**, qui représentent les choix possibles du système ;
- une **fonction objectif** linéaire, qui traduit le critère d'optimisation (minimisation ou maximisation) ;
- un ensemble de **contraintes linéaires**, qui définissent les conditions de validité et les limitations du problème étudié.

Sous forme générale, un problème de PLNE s'écrit :

$$\begin{aligned} &\max \text{ (ou } \min\text{)} && c^\top x \\ &\text{sous les contraintes} && Ax \leq b, \\ & && x \in \mathbb{Z}^n_{\geq 0}, \end{aligned}$$

où $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ et $b \in \mathbb{R}^m$ sont les données du problème, et $x \in \mathbb{Z}^n_{\geq 0}$ est le vecteur des variables de décision. Lorsque $x \in \{0,1\}^n$, on parle plus spécifiquement de programmation linéaire en variables binaires.

2.3.2 Intérêt et limites de la modélisation

La PLNE consiste à déterminer une solution optimale appartenant à un ensemble discret de solutions admissibles, défini par les contraintes du modèle. Cette approche permet de formaliser de manière rigoureuse des problèmes d'optimisation combinatoire en tenant compte simultanément de plusieurs contraintes et critères, ce qui en fait un outil largement utilisé en recherche opérationnelle pour traiter divers types de problèmes : affectation, planification, ordonnancement, localisation, etc.

L'utilisation de la PLNE présente notamment les avantages suivants :

— elle permet de représenter de manière structurée des problèmes complexes ;
— elle facilite la prise en compte de contraintes multiples et réalistes ;
— elle fournit, lorsque le solveur converge, des solutions optimales (ou bornées) directement exploitables dans des contextes décisionnels.

Remarque. *La résolution exacte d'un programme linéaire en nombres entiers est un problème NP-difficile dans le cas général, ce qui peut limiter son applicabilité à des instances de très grande taille. Toutefois, pour les tailles considérées dans le présent mémoire (de l'ordre d'une dizaine d'enseignants et environ quatorze modules), les solveurs modernes (CPLEX, Gurobi, GLPK, etc.) fournissent une solution optimale en un temps très raisonnable.*

Conclusion

Ce chapitre a permis de présenter et d'analyser les principales approches théoriques et méthodologiques utilisées pour résoudre le problème d'affectation dans les graphes bipartis.

Nous avons introduit l'algorithme de Hopcroft–Karp pour le cas non pondéré, l'algorithme Hongrois pour le cas pondéré, ainsi qu'une formulation en programmation linéaire en nombres entiers (PLNE) constituant un cadre de modélisation général et rigoureux.

L'ensemble de ces éléments forme le socle théorique nécessaire à la suite de ce travail, qui portera sur la modélisation détaillée et la mise en œuvre des différentes approches étudiées.

Chapitre 3

Modélisation du problème d'affectation

Introduction

Dans ce chapitre, nous nous intéressons à la modélisation du problème d'affectation des enseignants aux modules au sein du département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira.

Nous adoptons une approche fondée sur la théorie des graphes, et plus particulièrement sur les graphes bipartis, afin de représenter les relations entre les enseignants et les modules d'enseignement.

Cette modélisation permet de formaliser le problème en intégrant les différentes contraintes qui le caractérisent, notamment les compétences et habilitations des enseignants, les charges horaires maximales fonction du grade, les préférences individuelles, ainsi que les exigences pédagogiques propres à chaque module.

L'objectif est d'appliquer des méthodes d'optimisation afin de déterminer une affectation optimale des modules aux enseignants, tout en respectant l'ensemble des contraintes identifiées. Une telle approche permet d'automatiser le processus d'affectation, de limiter les conflits de planification et d'améliorer l'efficacité globale de l'organisation pédagogique.

3.1 Description du problème réel

3.1.1 Contexte

Dans les universités algériennes, l'organisation des enseignements constitue une tâche essentielle pour assurer le bon déroulement de la formation universitaire. À chaque semestre, les départements pédagogiques sont chargés de planifier et de répartir

les modules d'enseignement entre les enseignants disponibles.

Dans ce travail, nous nous intéressons particulièrement au cas du département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira. Comme dans les autres structures universitaires, cette organisation repose sur une répartition adéquate des modules entre les enseignants permanents, en tenant compte de plusieurs facteurs pédagogiques et administratifs. Cette affectation doit également respecter les contraintes liées au grade : certains modules exigent un niveau minimal de qualification, ce qui limite leur attribution aux enseignants de rang suffisant.

Chaque module est caractérisé par un volume horaire spécifique et peut inclure différentes formes d'enseignement : cours magistraux (CM), travaux dirigés (TD) et travaux pratiques (TP). L'affectation des modules doit donc être réalisée en fonction de plusieurs critères, notamment les spécialités scientifiques, le grade académique et la charge d'enseignement de chaque enseignant.

Cependant, dans la pratique, cette répartition est souvent effectuée de manière manuelle à l'aide de tableaux ou de feuilles de calcul, ce qui rend le processus long, complexe et difficile à optimiser, en particulier lorsque le nombre d'enseignants et de modules est élevé.

Dans ce contexte, le problème d'affectation des enseignants aux modules peut être formulé comme un problème d'optimisation combinatoire, visant à déterminer une répartition optimale tout en respectant un ensemble de contraintes pédagogiques et organisationnelles.

Les principales contraintes prises en compte sont les suivantes :

- **Habilitation :** l'enseignant doit posséder les compétences et qualifications nécessaires pour assurer un module donné.
- **Charge horaire :** le volume d'enseignement attribué à chaque enseignant doit respecter les limites maximales définies selon le grade, afin d'éviter toute surcharge ou sous-service.
- **Grade :** certains modules exigent un niveau de qualification minimal, ce qui impose des restrictions liées au statut de l'enseignant.
- **Préférences :** les souhaits des enseignants sont pris en compte dans la mesure du possible afin d'améliorer leur motivation et la qualité de l'enseignement.

La prise en compte de ces contraintes permet d'assurer une répartition cohérente et équilibrée des modules, tout en garantissant la qualité pédagogique et le respect des règles administratives.

3.1.2 Acteurs et données

Dans cette étude, nous avons analysé la répartition des enseignants sur les modules au sein du département de mathématiques de l'Université Akli Mohand Oulhadj de

Bouira.

Nous avons pris en compte 10 enseignants, en collectant pour chacun les informations suivantes : le nom, le prénom, le grade, la spécialité ainsi que la charge horaire maximale. Ces données ont été obtenues à partir des documents administratifs du département, en particulier des fichiers Excel internes.

Les informations sont résumées dans le tableau ci-dessous.

id_ens	**Nom**	**Prénom**	**Grade**	**Spécialité**	**Charge max**
E001	Djouder	Sofiane	MAA	Math. appliquées	09h
E002	Iftissan	El-ghani	MAA	Math. appliquées	09h
E003	Demmouche	Nacer	MCB	Probabilités, méthodes stochastiques	04h30
E004	Bekri	Houria	MAA	Analyse	09h
E005	Messoudene	Karima	MAA	Analyse	09h
E006	Hamid	Karim	MCB	Contrôle optimal	06h
E007	Ouidja	Daya	MCB	Contrôle optimal	09h
E008	Melouane	Nassima	MCB	EDP	09h
E009	Akkouche	Abderrahmane	MCA	Contrôle optimal	09h
E010	Birouche	Madjid	MAA	Statistique appliquée	09h

TABLE 3.1 – Liste des enseignants

Nous avons ensuite considéré 14 modules de différents niveaux, allant de la Licence jusqu'au Master. Pour chacun de ces modules, nous précisons les caractéristiques suivantes : l'intitulé, la filière, le semestre, le volume horaire (CM/TD/TP) ainsi que le grade minimum requis pour son enseignement. Ces informations sont issues de la maquette pédagogique officielle de la formation. Le tableau 3.2 présente ces données.

ID	Intitulé	Filière	Semestre	CM	TD	TP	Grade min
M001	Analyse 1	L1	S1	3h00	3h00	0	MCB
M002	Algèbre 1	L1	S1	1h30	1h30	0	MCB
M003	Probabilités-Statistique	L1	S2	1h30	1h30	0	MCB
M004	Analyse 3	L2	S3	3h00	3h00	0	MAA
M005	Analyse numérique 1	L2	S3	1h30	1h30	1h30	MAA
M006	Logique mathématique	L2	S3	1h30	1h30	0	MAA
M007	Algèbre 3	L2	S3	1h30	1h30	0	MAA
M008	Introduction à la topologie	L2	S3	3h00	3h00	0	MAA
M009	Géométrie	L2	S4	1h30	1h30	0	MAA
M010	Probabilités	L2	S4	1h30	1h30	0	MAA
M011	Analyse complexe	L2	S4	3h00	1h30	0	MAA
M012	Équations différentielles	L3	S5	3h00	1h30	0	MAA
M013	Optimisation sans contraintes	L3	S5	1h30	1h30	1h30	MAA
M014	Mesure et intégration	L3	S5	3h00	3h00	0	MAA

TABLE 3.2 – Liste des modules et leurs caractéristiques

Le tableau 3.3 présente la répartition des compétences des enseignants sur les différents modules. Chaque case contient une valeur binaire : la valeur 1 indique que l'enseignant est habilité à assurer le module, tandis que la valeur 0 indique le contraire. Ce tableau permet de visualiser rapidement les affectations possibles ; il sert de base pour évaluer l'adéquation et pour formuler les contraintes du modèle mathématique présenté plus loin.

ID	M001	M002	M003	M004	M005	M006	M007	M008	M009	M010	M011	M012	M013	M014
E001	1	1	1	0	0	0	0	0	0	1	0	0	1	0
E002	1	1	1	0	1	0	0	0	0	1	0	0	1	0
E003	1	1	1	1	0	0	0	0	0	1	1	0	0	1
E004	1	1	1	1	1	0	0	1	1	0	1	0	0	0
E005	1	1	0	1	1	0	1	1	0	0	1	0	0	0
E006	1	1	0	1	1	0	1	1	0	0	1	1	1	0
E007	1	1	0	1	1	0	1	1	0	0	1	1	1	0
E008	1	1	0	1	1	0	1	1	1	0	1	1	0	0
E009	1	1	1	0	1	1	0	0	0	1	0	1	1	0
E010	1	1	1	1	1	1	0	1	1	1	1	1	0	1

TABLE 3.3 – Tableau d'habilitation des enseignants aux modules

Pour assurer une répartition pertinente des enseignants sur les modules, il est nécessaire de tenir compte non seulement de leurs compétences, mais également de leurs préférences pédagogiques. Le tableau 3.4 ci-dessous présente, à titre d'illustration, un extrait de ces préférences pour quelques enseignants, sur une échelle de 0 à 10 : la valeur 0 traduit une absence d'intérêt, tandis que la valeur 10 exprime une motivation maximale. L'intégralité du tableau est présentée en annexe (tableau A.1).

Ces données fournissent une vue synthétique des affinités de chaque enseignant et constituent une base pertinente pour aboutir à une affectation équilibrée, conciliant les exigences institutionnelles et les souhaits individuels en vue de garantir un enseignement de qualité.

Enseignant	Module	Préférence
E001	M001	5
E001	M002	6
E001	M003	7
E001	M010	7
E001	M013	9
E002	M001	7
E002	M002	6
E002	M003	9
E002	M010	8
E002	M013	6
E003	M001	7
E003	M002	6
E003	M003	10
E003	M004	7
E003	M010	10
E003	M011	5
E003	M014	9

TABLE 3.4 – Préférences des enseignants pour les modules (extrait)

3.2 Modélisation par graphe biparti d'habilitation

Pour modéliser l'affectation des enseignants aux modules, nous adoptons la représentation par graphe biparti. Un tel graphe sépare ses sommets en deux classes disjointes, et toute arête relie un sommet de l'une à un sommet de l'autre, jamais deux sommets d'une même classe.

3.2.1 Ensembles de sommets

— **Enseignants ($\mathcal{E}$).** Cette classe regroupe l'ensemble des enseignants disponibles du département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira. Chaque sommet $e \in \mathcal{E}$ représente un enseignant donné.

— **Modules ($\mathcal{M}$).** Cette classe regroupe l'ensemble des modules à affecter. Chaque sommet $m \in \mathcal{M}$ correspond à un module du programme pédagogique.

Les ensembles $\mathcal{E}$ et $\mathcal{M}$ sont disjoints : $\mathcal{E} \cap \mathcal{M} = \emptyset$.

3.2.2 Ensemble des arêtes

Les arêtes de l'ensemble $A \subseteq \mathcal{E} \times \mathcal{M}$ traduisent les habilitations. Une arête $(e, m) \in A$ exprime le fait que l'enseignant e est apte à prendre en charge le module m, conformément aux critères pédagogiques et à son parcours.

3.2.3 Interprétation du modèle

Ce modèle exclut tout lien direct entre enseignants ou entre modules : seules les relations croisées entre les deux classes sont représentées. Il fournit ainsi le socle structurel auquel sont ensuite greffées les contraintes opérationnelles du problème (charge horaire, grade, préférences, etc.).

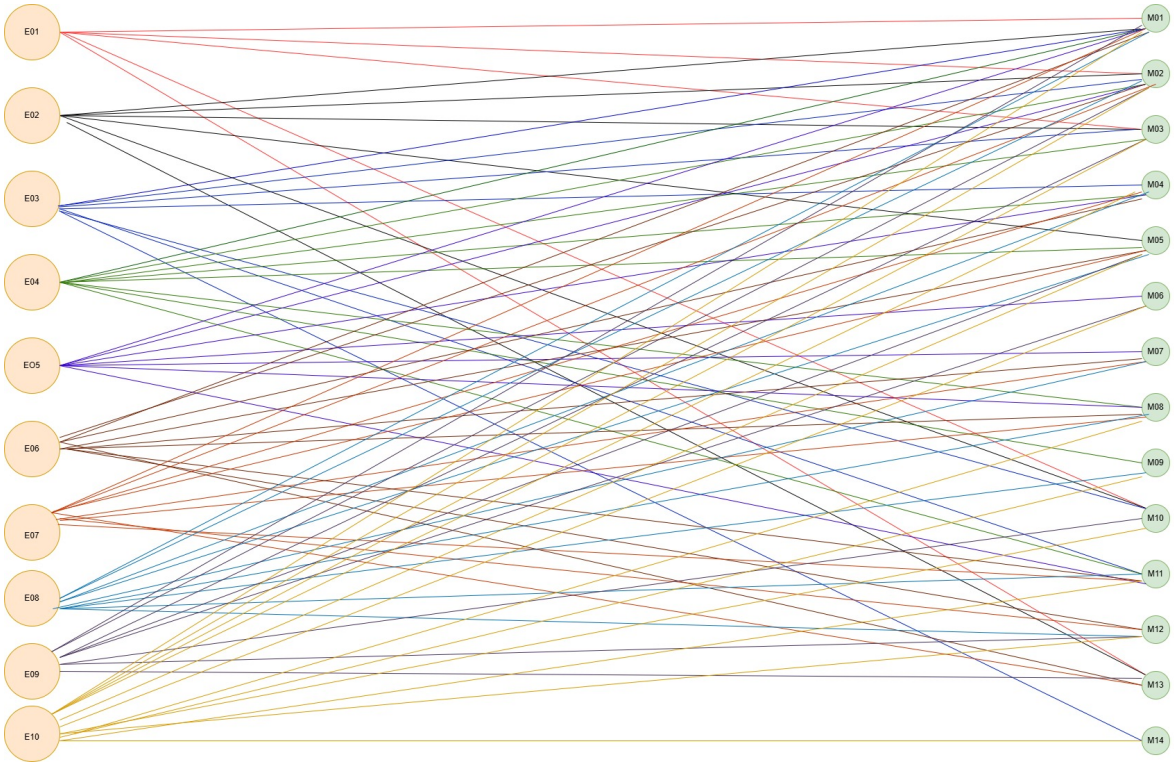


FIGURE 3.1 – Graphe biparti d'affectation des enseignants aux modules

3.2.4 Graphe pondéré et fonction de score

Dans le contexte de l'affectation des enseignants aux modules, il est essentiel de disposer d'un critère objectif pour évaluer l'adéquation de chaque enseignant à chaque module. C'est dans ce but que nous proposons une fonction de score d'adéquation, qui mesure le degré de compatibilité entre un enseignant et un module.

Cette fonction de score pondérée constitue un outil clé pour optimiser la qualité des affectations. Elle allie objectivité, flexibilité et simplicité de mise en œuvre, et s'intègre aisément dans un système automatisé d'affectation, garantissant une meilleure correspondance entre les compétences des enseignants et les exigences des modules.

Formule mathématique de la fonction de score

Pour chaque couple enseignant-module (e_i, m_j), le score d'adéquation C_{ij} est défini par la combinaison linéaire suivante :

$$C_{ij} = \alpha \cdot s_{\text{spec}}(e_i, m_j) + \beta \cdot s_{\text{grade}}(e_i, m_j) + \gamma \cdot s_{\text{pref}}(e_i, m_j),$$

où :

- $s_{\text{spec}}(e_i, m_j) \in [0, 10]$: adéquation entre la spécialité de l'enseignant et le domaine du module ;
- $s_{\text{grade}}(e_i, m_j) \in [0, 10]$: adéquation du grade de l'enseignant avec le niveau du module ;
- $s_{\text{pref}}(e_i, m_j) \in [0, 10]$: préférence déclarée de l'enseignant pour le module.

Les poids α, β et γ satisfont la contrainte de normalisation :

$$\alpha + \beta + \gamma = 1.$$

Dans la suite, nous fixerons par exemple $\alpha = 0.5$, $\beta = 0.3$ et $\gamma = 0.2$.

Importance des critères

- **Spécialité (s_{spec}) :** critère dominant, il garantit une qualité d'enseignement supérieure grâce à la maîtrise des concepts par l'enseignant.
- **Grade (s_{grade}) :** il reflète l'expérience académique et la capacité à adapter le contenu au niveau du module.
- **Préférences (s_{pref}) :** elles favorisent l'efficacité pédagogique en tenant compte de l'intérêt de l'enseignant, mais leur poids reste modéré afin de limiter leur influence sur la décision finale.

Intégration dans le graphe pondéré

Grâce à cette fonction de score, le graphe modélisant les enseignants et les modules devient pondéré, chaque arête (e_i, m_j) portant la valeur C_{ij}. Cela permet :

— de comparer objectivement les différentes options d'affectation ;
— de repérer les conflits ou les modules problématiques ;
— d'appliquer des algorithmes d'optimisation afin d'obtenir une répartition optimale des enseignants.

Le tableau 3.5 ci-dessous illustre, sur un extrait, la méthode de calcul du score d'adéquation à partir de la fonction pondérée. Pour clarifier la démarche, nous nous limitons à trois enseignants ($E001$, $E002$, $E003$). La méthode s'applique de manière analogue à l'ensemble des enseignants et des modules ; le tableau complet est présenté en annexe (tableau A.2).

id_ens	**id_mod**	s_{spec}	s_{grade}	s_{pref}	**Score total**
E001	M001	7	6	5	6.3
E001	M002	7	6	6	6.5
E001	M003	8	6	7	7.2
E001	M010	8	8	7	7.8
E001	M013	10	8	9	9.2
E002	M001	7	6	7	6.7
E002	M002	7	6	6	6.5
E002	M003	8	6	9	7.6
E002	M010	8	8	8	8.0
E002	M013	10	8	6	8.6
E003	M001	6	8	7	6.8
E003	M002	6	8	6	6.6
E003	M003	10	8	10	9.4
E003	M004	6	9	7	7.1
E003	M010	10	9	10	9.7
E003	M011	6	9	5	6.7
E003	M014	7	9	9	8.0

TABLE 3.5 – Scores d'adéquation enseignants-modules (extrait)

À partir du tableau 3.5, on construit le graphe pondéré illustré par la figure 3.2.

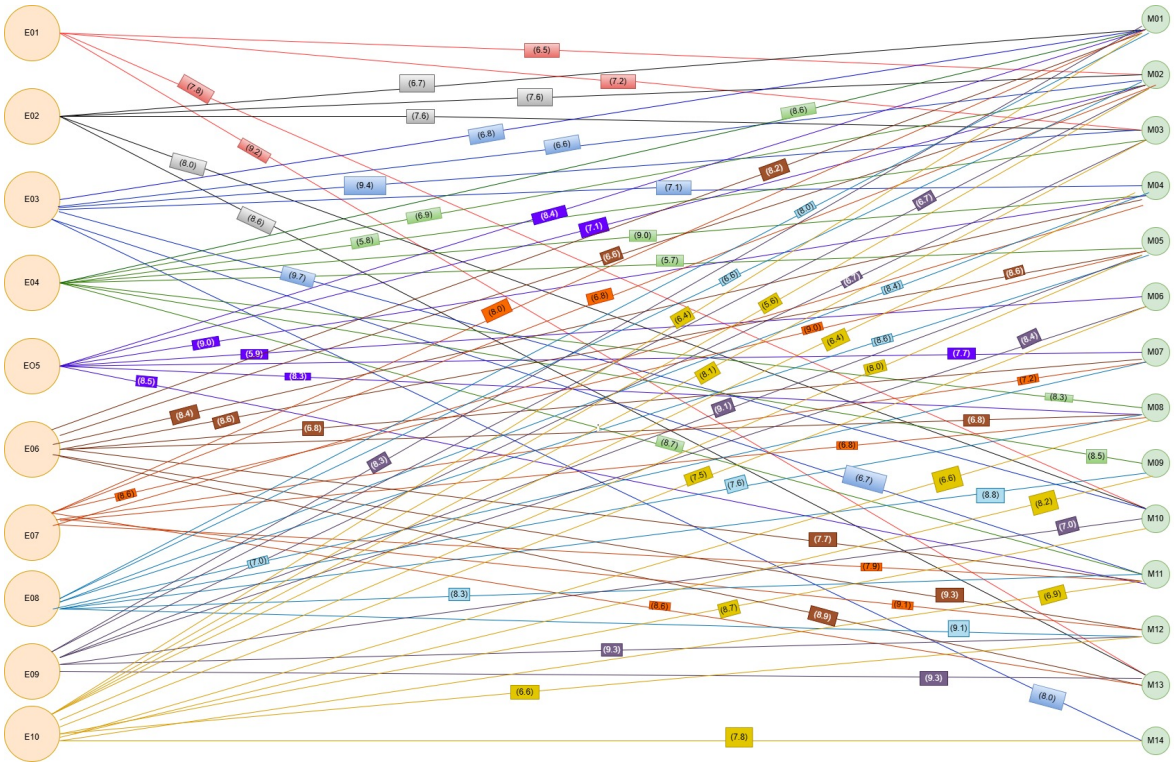


FIGURE 3.2 – Graphe pondéré d'affectation des enseignants aux modules

3.3 Formulation mathématique complète

Avant d'entrer dans les détails de la formulation mathématique, nous précisons la terminologie et introduisons les ensembles afin de rendre la présentation du modèle plus claire. Nous définissons d'abord les principaux paramètres et variables, puis l'objectif à optimiser et, enfin, les contraintes auxquelles cet objectif est soumis.

3.3.1 Paramètres

Les paramètres du modèle représentent les données d'entrée indispensables à sa formulation. Ils traduisent les caractéristiques des enseignants et des modules ainsi que les contraintes institutionnelles. Leur définition permet de structurer le problème et d'assurer une modélisation à la fois cohérente, réaliste et exploitable dans un cadre d'optimisation.

- n : nombre d'enseignants disponibles ;
- k : nombre de modules à affecter ;
- $c_{ij} \in \mathbb{R}_{\geq 0}$: score d'adéquation entre l'enseignant i et le module j ;
- $h_{ij} \in \{0, 1\}$: indicateur d'habilitation ($h_{ij} = 1$ si l'enseignant i peut assurer le module j, 0 sinon) ;
- $CH_i^{\max}$: charge horaire maximale autorisée pour l'enseignant i ;
- VH_j : volume horaire global du module j ;

— G_i : grade de l'enseignant i ;
— $G_j^{\min}$: grade minimum requis pour assurer le module j.

3.3.2 Variables de décision

Pour modéliser le problème d'affectation des enseignants aux modules, nous introduisons des variables de décision binaires, qui indiquent simplement si un enseignant i est affecté à un module j ou non :

$$x_{ij} = \begin{cases} 1 & \text{si l'enseignant } i \text{ est affecté au module } j, \\ 0 & \text{sinon.} \end{cases}$$

3.3.3 Fonction objectif du modèle

L'objectif principal de ce modèle est de maximiser la qualité globale des affectations enseignants-modules. Pour chaque couple enseignant-module, un score d'adéquation est calculé en tenant compte de la spécialité, du grade, des préférences et, plus généralement, de l'ensemble des éléments rendant l'affectation cohérente.

Formellement, la fonction objectif s'écrit :

$$\max \ Z = \sum_{i=1}^{n} \sum_{j=1}^{k} c_{ij} \cdot x_{ij}.$$

Cet objectif favorise les choix les plus pertinents sur les plans pédagogique et organisationnel. Il est bien entendu encadré par un ensemble de contraintes opérationnelles—habilitations, charges horaires, grades—qui garantissent la faisabilité de la solution dans un cadre réel.

3.3.4 Contraintes du modèle

Afin de garantir la faisabilité et la pertinence des affectations proposées, le modèle d'optimisation est encadré par un ensemble de contraintes opérationnelles et institutionnelles, directement inspirées des règles pédagogiques et administratives en vigueur.

Contrainte (C1) : unicité de l'affectation pour chaque enseignant. Cette contrainte impose qu'un enseignant ne soit affecté qu'à un seul module au plus :

$$\sum_{j=1}^{k} x_{ij} \leq 1, \qquad \forall\, i \in \{1, \ldots, n\}. \tag{3.1}$$

Elle garantit qu'un même enseignant ne peut être responsable de plusieurs modules dans cette première version du modèle.

Contrainte (C2) : couverture des modules. Chaque module doit être pris en charge par exactement un enseignant :

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall\, j \in \{1, \ldots, k\}. \tag{3.2}$$

Aucun module n'est ainsi laissé sans responsable, et aucune affectation multiple n'est autorisée pour un même module.

Contrainte (C3) : respect des habilitations. Cette contrainte traduit le respect des habilitations pédagogiques :

$$x_{ij} \leq h_{ij}, \qquad \forall\, i, j. \tag{3.3}$$

Un enseignant i ne peut être affecté au module j que s'il y est habilité ($h_{ij} = 1$).

Contrainte (C4) : charge horaire maximale. Le volume horaire total attribué à chaque enseignant ne doit pas excéder sa charge maximale :

$$\sum_{j=1}^{k} VH_j \cdot x_{ij} \leq CH_i^{\max}, \qquad \forall\, i. \tag{3.4}$$

Cette contrainte permet d'éviter toute surcharge de travail.

Contrainte (C5) : contrainte de grade. Le grade minimum requis pour enseigner un module doit être respecté :

$$x_{ij} = 0 \quad \text{si } G_i < G_j^{\min}, \qquad \forall\, i, j. \tag{3.5}$$

Cette contrainte exclut toute affectation ne satisfaisant pas les exigences académiques.

Contrainte (C6) : nature des variables. Les variables de décision sont binaires :

$$x_{ij} \in \{0, 1\}, \qquad \forall\, i, j. \tag{3.6}$$

Cette contrainte traduit le caractère discret de la décision : un enseignant est soit affecté à un module, soit ne l'est pas.

3.4 Extension : affectation multi-modules

En pratique, un enseignant n'est presque jamais limité à un seul module : il peut intervenir dans plusieurs unités d'enseignement au cours d'un même semestre. Pour

refléter cette situation plus fidèlement, on assouplit la contrainte (C1) en autorisant l'affectation d'un même enseignant à plusieurs modules, à condition de respecter sa charge horaire maximale (contrainte (C4)).

Ainsi, dans cette version étendue du modèle, la contrainte (C1) est purement et simplement supprimée, et la limitation effective de l'activité de l'enseignant est portée par la contrainte de charge horaire :

$$\sum_{j=1}^{k} VH_j \cdot x_{ij} \leq CH_i^{\max}, \qquad \forall\, i.$$

Autrement dit, la somme des heures associées aux modules attribués à un enseignant ne doit jamais dépasser sa capacité maximale, mais le nombre de modules par enseignant n'est plus borné explicitement.

3.5 Collecte et préparation des données

3.5.1 Collecte des données

Dans le cadre de ce travail, les données ont été structurées sous forme de fichiers au format CSV afin de faciliter leur exploitation par les différents modèles d'affectation développés. Ces données décrivent les principales entités du problème : les enseignants, les modules et les habilitations.

3.5.2 Description des fichiers de données

Le fichier `enseignants.csv` regroupe les informations relatives aux enseignants : identifiant, nom, prénom, grade, spécialité et charge horaire maximale. Il représente les ressources humaines disponibles dans le système.

Extrait du fichier `enseignants.csv` :

```
id_ens ; nom ; prenom ; grade ; specialite ; charge_max
E001 ; Djouder ; Sofiane ; MAA ; Mathematiques appliquees ; 09h
E002 ; Iftissan ; El-ghani ; MAA ; Mathematiques appliquees ; 09h
```

Le fichier `modules.csv` contient les caractéristiques des modules à enseigner : intitulé, filière, semestre, volumes horaires (CM/TD/TP) et grade minimum requis.

Extrait du fichier `modules.csv` :

```
id_mod ; intitule ; filiere ; semestre ; vh_cours ; vh_td ; vh_tp ;
grade_min
M001 ; Analyse 1 ; L1 ; S1 ; 3h00 ; 3h00 ; 0 ; MCB
```

```
M002 ; Algebre 1 ; L1 ; S1 ; 1h30 ; 1h30 ; 0 ; MCB
M003 ; Proba-Stat ; L1 ; S2 ; 1h30 ; 1h30 ; 0 ; MCB
```

Le fichier `habilitations.csv` définit la relation entre les enseignants et les modules. Chaque couple est associé à plusieurs scores (spécialité, grade, préférence et score global), permettant d'évaluer la compatibilité entre un enseignant et un module.

Extrait du fichier `habilitations.csv` :

```
id_ens ; id_mod ; spec ; grade ; pref ; total
E001 ; M001 ; 7 ; 6 ; 5 ; 6.3
E001 ; M002 ; 7 ; 6 ; 6 ; 6.5
E001 ; M003 ; 8 ; 6 ; 7 ; 7.2
```

3.5.3 Préparation des données

Après leur collecte, les données sont structurées et transformées afin d'être exploitables par les modèles proposés. Elles sont organisées sous forme de matrices et de graphes bipartis, ce qui facilite l'application des algorithmes de couplage et d'optimisation développés dans ce travail.

Cette étape constitue une phase essentielle du processus, garantissant la cohérence des données et la fiabilité des résultats obtenus lors des différentes expérimentations.

Conclusion

Dans ce chapitre, nous avons présenté une modélisation complète du problème d'affectation des enseignants aux modules. Le problème a été formulé comme un programme linéaire en nombres entiers (PLNE) sur un graphe biparti pondéré. Le modèle proposé intègre l'ensemble des contraintes essentielles : disponibilité des enseignants, spécialisation et expertise, équilibre des charges horaires, et possibilité pour un enseignant de prendre en charge plusieurs modules sans dépasser sa charge maximale. Les affectations obtenues sont ainsi non seulement réalisables, mais également pertinentes du point de vue pédagogique.

L'approche par PLNE sur graphe biparti pondéré offre par ailleurs plusieurs avantages pratiques : elle permet une mise en œuvre directe des algorithmes d'optimisation, autorise l'ajout aisé de nouvelles contraintes, et offre une flexibilité appréciable pour adapter les affectations aux besoins du département.

Le chapitre suivant abordera l'implémentation en Python ainsi que les résultats expérimentaux permettant de valider la démarche.

Chapitre 4

Implémentation et résultats expérimentaux

Introduction

Ce chapitre est consacré à la mise en œuvre concrète des méthodes et algorithmes présentés précédemment. Le langage Python a été retenu comme outil principal pour l'implémentation des solutions et l'analyse des résultats. L'objectif est de relier les développements théoriques à la pratique au moyen d'un modèle complet, illustrant le traitement des données et l'application efficace des algorithmes étudiés.

Nous présentons ensuite les algorithmes retenus l'algorithme de Hopcroft–Karp et l'algorithme Hongrois en détaillant leur fonctionnement et en justifiant leur pertinence pour les problèmes d'affectation et de couplage considérés.

Enfin, nous exposons les résultats des expérimentations menées sur les données réelles du département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira. L'analyse porte sur la qualité des solutions, le temps d'exécution et la précision des affectations, et met en évidence l'utilité pratique du système développé pour appuyer la planification pédagogique et la prise de décision académique.

4.1 Architecture de la solution

4.1.1 Vue d'ensemble

Une architecture modulaire a été retenue pour la conception du système. L'application est ainsi divisée en plusieurs modules indépendants, chacun dédié à une fonction précise.

Cette structure rend le code plus clair, facilite sa compréhension et sa maintenance, et permet d'introduire des évolutions ou des modifications sans impacter les autres com-

posants. Le tableau 4.1 présente la répartition des modules ainsi que leurs principales caractéristiques.

Module	**Objectif principal**	**Fonctionnalités clés**	**Bibliothèques**
`Hopcroft_karp.py`	Couplage maximum dans un graphe biparti	Recherche d'un couplage optimal via BFS et DFS (chaînes augmentantes)	`collections`
`algorithme_Hongrois.py`	Affectation optimale pondérée	Minimisation de la matrice de coûts pour obtenir la meilleure affectation	NumPy, SciPy
`interface.py`	Interface utilisateur	Interaction avec l'utilisateur, exécution des algorithmes et affichage des résultats	Tkinter
`main.py`	Coordination du système	Chargement des données CSV et orchestration des modules	Modules Python internes

TABLE 4.1 – Architecture modulaire du système développé

4.2 Implémentation des algorithmes

4.2.1 Exploitation des données

Dans ce travail, le graphe biparti a été représenté visuellement dans l'application présentée au chapitre 3. Cette représentation a été réalisée à l'aide de l'outil en ligne https://app.diagrams.net/, afin de faciliter la compréhension des relations entre enseignants et modules. Dans la présente section, les données issues des fichiers CSV sont exploitées en Python pour appliquer les algorithmes d'affectation, sans représentation graphique supplémentaire.

4.2.2 Algorithme de Hopcroft–Karp

Nous présentons ici une description générale de la phase de préparation des données ainsi qu'une version simplifiée de l'algorithme de couplage utilisé. Le programme charge les données à partir des fichiers CSV, construit les relations entre enseignants et modules à partir des habilitations, puis détermine une affectation des modules disponibles aux enseignants compatibles.

Le programme procède ensuite à une répartition simple en attribuant les modules disponibles aux enseignants compatibles, dans le respect des contraintes définies. Il affiche enfin la répartition obtenue ainsi que le nombre total d'affectations.

Pour des raisons de clarté et de lisibilité, l'ensemble du code source de cette implémentation est présenté en annexe (Annexe C).

4.2.3 Algorithme Hongrois (via SciPy)

Nous présentons dans cette partie une implémentation de l'algorithme Hongrois appliqué au problème d'affectation des enseignants aux modules. L'objectif est d'obtenir une répartition équilibrée respectant une capacité maximale par enseignant et maximisant le score global d'affectation.

Le programme commence par une affectation initiale simple, qui assure une répartition de base. Les affectations restantes sont ensuite optimisées à l'aide d'une matrice de coûts et de l'algorithme Hongrois.

Le problème de maximisation des scores est reformulé comme un problème de minimisation en utilisant la matrice de coût négative $C_{\text{neg}} = -C$, ce qui permet de déterminer l'affectation optimale conformément à la convention de la fonction `linear_sum_assignment` de `SciPy`.

Pour des raisons de clarté et de lisibilité, l'ensemble du code source de cette implémentation est présenté en annexe (Annexe D).

4.3 Visualisation des résultats

Dans cette section, nous présentons les résultats obtenus après l'exécution des programmes développés en Python pour l'application des algorithmes de Hopcroft–Karp et Hongrois. L'objectif est d'exposer les affectations obtenues entre les enseignants et les modules, ainsi que la représentation graphique des solutions afin d'en faciliter la lecture.

4.3.1 Résultats de l'algorithme de Hopcroft–Karp

Après l'exécution de l'algorithme de Hopcroft–Karp, la répartition des modules entre les enseignants présentée dans le tableau 4.2 a été obtenue.

Enseignant	**Modules affectés**
E001	M001, M013
E002	M002
E003	M003, M011
E004	M004
E005	M005
E006	M007, M012
E007	M008
E008	M009
E009	M006
E010	M010, M014

TABLE 4.2 – Répartition obtenue par l'algorithme de Hopcroft–Karp

Le nombre total d'affectations obtenues est de 14, ce qui montre que tous les modules ont été attribués avec succès.

La figure 4.1 présente la représentation graphique de la répartition obtenue.

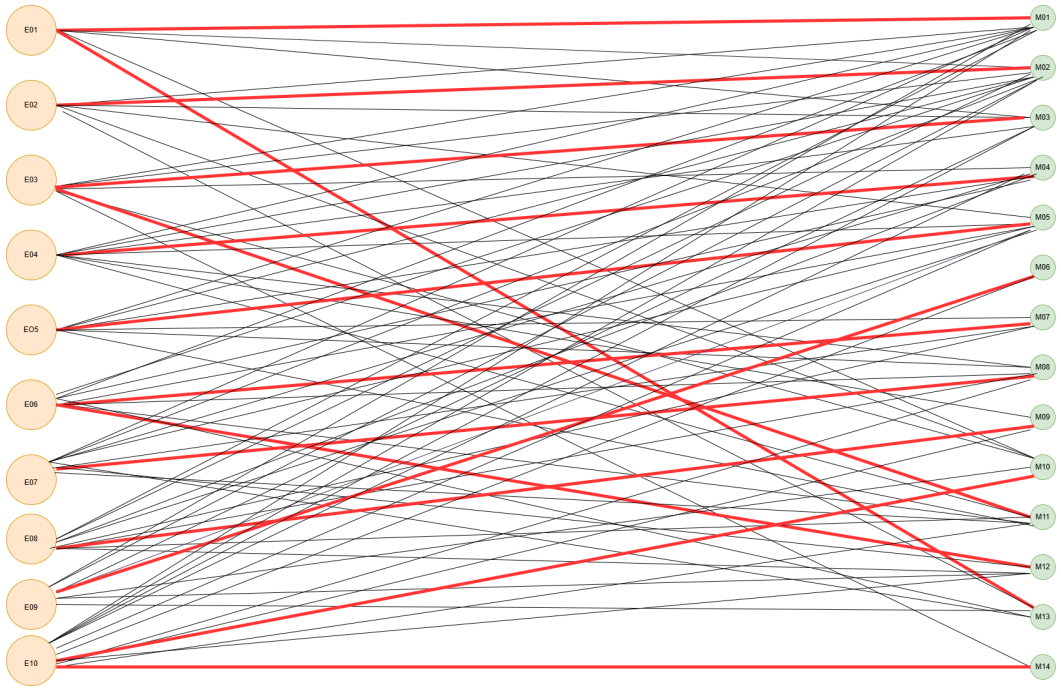


FIGURE 4.1 – Couplage maximum obtenu par l'algorithme de Hopcroft–Karp

Les résultats obtenus montrent que l'algorithme produit une affectation cohérente entre enseignants et modules. La répartition est équilibrée et exploite efficacement les ressources disponibles.

4.3.2 Résultats de l'algorithme Hongrois

L'application de l'algorithme Hongrois a permis d'obtenir la répartition optimale des enseignants aux modules présentée dans le tableau 4.3.

Enseignant	**Module**	**Score**
E001	M013	9.2
E002	M010	8.0
E003	M003	9.4
E004	M001	8.6
E005	M011	8.5
E006	M012	9.3
E007	M004	9.0
E008	M009	8.8
E009	M005	9.1
E010	M014	7.8
E004	M008	8.3
E005	M007	7.7
E005	M002	7.1
E009	M006	8.4

TABLE 4.3 – Résultats de l'algorithme Hongrois

Le score total obtenu est de 119,19

La figure 4.2 présente la visualisation graphique de la solution obtenue.

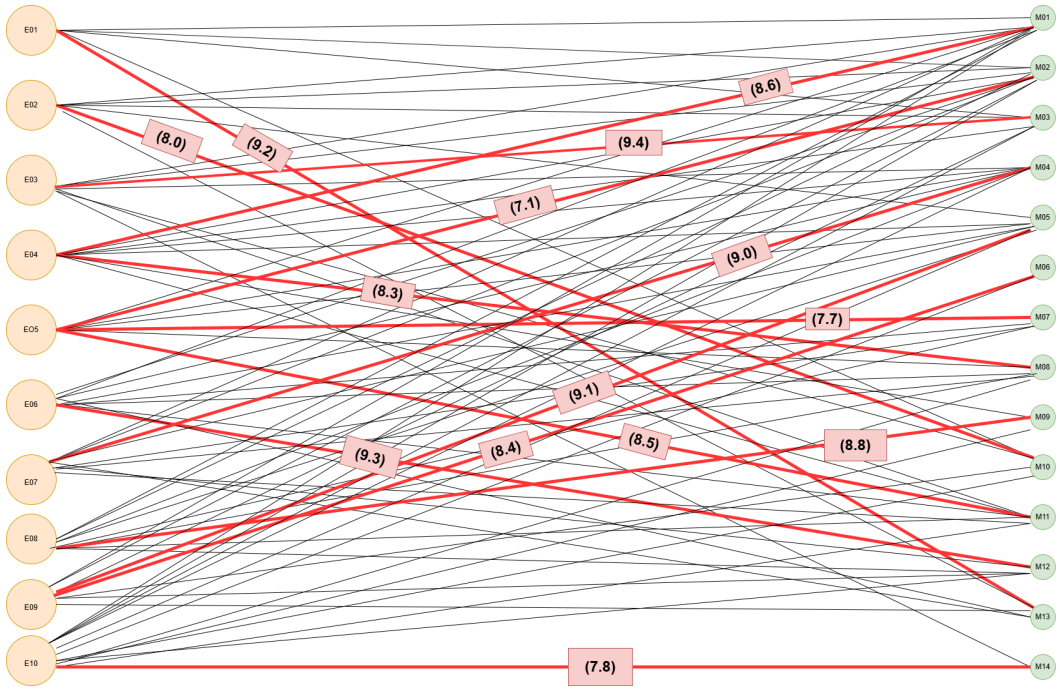


FIGURE 4.2 – Affectation optimale finale obtenue par l'algorithme Hongrois

Cette représentation graphique permet de visualiser de manière claire la solution d'affectation obtenue, en mettant en évidence la répartition optimale des enseignants aux modules.

4.3.3 Résultats et analyse de la solution obtenue par GLPK

La résolution du modèle d'optimisation, effectuée à l'aide du solveur GLPK, a permis d'obtenir une solution optimale satisfaisant l'ensemble des contraintes définies dans le modèle.

Le modèle traité comporte 140 variables de décision et 175 contraintes, ce qui reflète la taille et la complexité du problème d'affectation. La valeur de la fonction objectif obtenue est égale à 119,1, correspondant au score global maximal atteint par la solution optimale.

L'analyse des résultats met en évidence les points suivants :

— l'ensemble des modules a été affecté, garantissant une couverture complète du programme pédagogique ;

— les contraintes d'habilitation des enseignants ont été strictement respectées ;

— les charges horaires respectent les limites fixées, garantissant un équilibre dans la répartition du travail ;

— l'affectation de certains enseignants à plusieurs modules permet une utilisation optimale des ressources humaines disponibles.

Le code d'implémentation associé est présenté en annexe (Annexe E).

Le tableau 4.4 présente l'affectation finale des enseignants aux modules ainsi que le score associé à chaque enseignant, calculé en fonction des préférences et des coefficients du modèle.

Enseignant	**Modules affectés**	**Score obtenu**
E001	M013	9.2
E002	M002	6.5
E003	M003	9.4
E004	M001	8.6
E005	M007, M008	16.0
E006	M012	9.3
E007	M004	9.0
E008	M009,M011	17.1
E009	M005, M006	17.5
E010	M010, M014	16.5

TABLE 4.4 – Solution optimale : affectation des enseignants aux modules avec score associé

4.4 Analyse comparative des approches d'affectation

Cette section présente une étude comparative des trois méthodes utilisées dans ce chapitre pour résoudre le problème d'affectation des enseignants aux modules : l'algorithme de Hopcroft–Karp, l'algorithme Hongrois et le modèle de programmation linéaire en nombres entiers (PLNE) résolu à l'aide du solveur GLPK.

L'objectif est d'évaluer leurs performances sur une même instance réelle, afin d'analyser les différences en termes de qualité des solutions, de respect des contraintes et d'efficacité d'exécution. Cette comparaison fournit une évaluation objective et représentative des approches étudiées.

4.4.1 Tableau comparatif

Le tableau 4.5 résume les principales caractéristiques des trois approches étudiées selon plusieurs critères de comparaison.

Critère	Hopcroft–Karp	Algorithme Hongrois	GLPK (modèle PLNE)
Qualité de la solution	**Faible :** méthode focalisée uniquement sur le nombre d'affectations.	**Bonne :** prise en compte des préférences pour améliorer les affectations.	**Optimale :** solution optimale sous l'ensemble des contraintes.
Score (satisfaction)	**Faible :** ne tient compte ni des poids ni des préférences.	**Très élevé :** Meilleure performance observée.	**Très élevé :** Résultat très proche de la meilleure solution.
Respect des contraintes	Limité aux contraintes de couplage biparti.	**Partiel :** ne prend pas en compte toutes les contraintes du modèle.	**Complet :** gère l'ensemble des contraintes du problème.
Temps d'exécution	**Très rapide :** le plus performant en termes de vitesse.	**Rapide :** adapté aux instances de taille moyenne.	**Modéré :** temps d'exécution plus élevé en raison de la complexité du modèle.

TABLE 4.5 – Tableau comparatif des trois approches d'affectation

4.4.2 Analyse et discussion des résultats

À partir des résultats expérimentaux présentés dans les tableaux 4.2, 4.3 et 4.4, la comparaison des trois approches conduit aux observations suivantes.

— **Algorithme de Hopcroft–Karp.** Cette approche se distingue par sa rapidité d'exécution et sa faible complexité algorithmique. Elle permet de maximiser le nombre d'affectations, mais ne tient compte ni des préférences ni des contraintes qualitatives, ce qui restreint son utilisation aux problèmes structurels simples. Le niveau global de satisfaction obtenu demeure inférieur à celui des autres approches étudiées.

— **Algorithme Hongrois.** Cette méthode améliore la qualité des affectations en intégrant une fonction de coût fondée sur les préférences. Elle produit une solution de très grande qualité et obtient les meilleurs résultats en termes de score. Toutefois, elle reste limitée lorsqu'il s'agit de représenter des contraintes complexes liées au contexte réel.

— **Modèle PLNE résolu par GLPK.** Cette approche constitue la solution la plus complète. Elle permet de modéliser l'ensemble des contraintes du problème et de garantir une solution optimale par rapport au modèle formulé. Malgré un temps de calcul légèrement plus élevé, elle demeure adaptée à la taille de l'instance étudiée et représente l'approche la plus pertinente dans un contexte réel.

Cette analyse montre que le choix d'une méthode d'affectation ne repose pas uniquement sur la performance algorithmique, mais également sur la capacité à modéliser et à respecter les contraintes du problème.

Ainsi, bien que l'algorithme de Hopcroft–Karp se distingue par sa rapidité et que l'algorithme Hongrois obtienne les meilleurs résultats en termes de score, le modèle

PLNE résolu par GLPK apparaît comme la solution la plus adaptée grâce à sa capacité à intégrer l'ensemble des contraintes et à fournir une modélisation plus réaliste du problème étudié.

4.5 Interface utilisateur

Dans le cadre de ce travail, une interface graphique a été développée à l'aide de la bibliothèque `Tkinter` du langage Python, afin de faciliter l'utilisation des méthodes d'affectation proposées. Cette interface a été conçue de manière à être simple et intuitive, permettant à l'utilisateur d'interagir aisément avec le système.

L'application développée permet notamment :

— de charger les données à partir d'un fichier CSV contenant les informations relatives aux enseignants, aux modules et aux scores d'affectation ;

— de sélectionner l'algorithme d'affectation à utiliser : l'algorithme de **Hopcroft–Karp** pour le cas non pondéré ou l'algorithme **Hongrois** pour le cas pondéré ;

— de lancer le processus de calcul pour générer automatiquement une solution d'affectation ;

— d'afficher les résultats sous forme de tableau, ce qui en facilite la lecture et l'interprétation ;

— de visualiser le graphe biparti correspondant à la solution obtenue ;

— d'exporter les résultats au format CSV pour une utilisation ultérieure.

Cette interface permet ainsi de mettre en œuvre concrètement les différentes approches étudiées, tout en offrant un outil pratique pour analyser les résultats obtenus.

La figure 4.3 présente une illustration de l'interface développée.

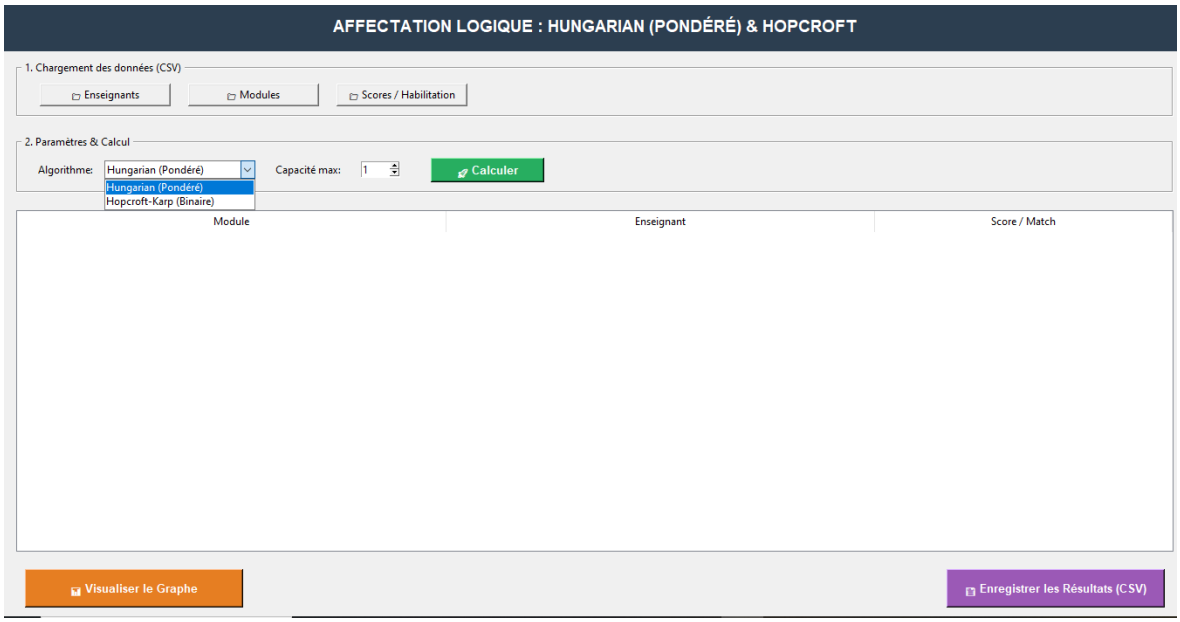


FIGURE 4.3 – Interface graphique de l'application d'affectation

Le code source complet de l'application est présenté en annexe (Annexe F) afin de permettre la reproduction des résultats.

Conclusion

Dans ce chapitre, nous avons concrétisé les développements théoriques précédents en exploitant des données réelles issues de fichiers CSV pour résoudre le problème d'affectation des enseignants aux modules.

Trois approches ont été appliquées : l'algorithme de Hopcroft–Karp, l'algorithme Hongrois et le modèle PLNE résolu par GLPK.

Les résultats obtenus mettent en évidence les avantages spécifiques de chaque méthode. La résolution par GLPK fournit la solution la plus complète et la mieux adaptée aux contraintes du problème.

Ce chapitre valide ainsi les approches étudiées et confirme leur efficacité dans le traitement du problème d'affectation.

Conclusion générale

Ce mémoire a été consacré à l'étude du problème d'affectation des enseignants aux modules d'enseignement, en s'appuyant sur les outils de la théorie des graphes et de l'optimisation combinatoire. L'objectif principal était d'aboutir à une affectation cohérente et efficace, respectant les contraintes pédagogiques et administratives propres au contexte universitaire.

Les expérimentations réalisées constituent l'apport principal de ce travail. Elles ont mis en évidence des résultats clairs quant aux performances des différentes approches étudiées. L'algorithme de Hopcroft–Karp permet d'obtenir rapidement un couplage maximum, ce qui le rend particulièrement adapté aux situations nécessitant une solution immédiate. Cette approche reste néanmoins limitée en termes de qualité globale, car elle ne prend en compte aucun critère qualitatif d'optimisation.

En revanche, l'algorithme Hongrois et la modélisation en programmation linéaire en nombres entiers (PLNE) ont permis d'obtenir des solutions nettement plus pertinentes, assurant une meilleure adéquation entre les enseignants et les modules. Les résultats obtenus montrent que ces méthodes améliorent significativement la qualité des affectations, au prix d'un temps de calcul plus important. La résolution du modèle PLNE à l'aide du solveur GLPK a en particulier confirmé la capacité de cette formulation à intégrer simultanément l'ensemble des contraintes du problème (habilitations, charges horaires, grades, préférences) et à fournir une solution optimale, ce qui en fait l'approche la plus adaptée au contexte étudié.

La comparaison des différentes méthodes met ainsi en évidence un compromis entre rapidité et qualité de solution, chaque approche étant adaptée à un contexte d'utilisation spécifique. L'implémentation en Python et le développement d'une interface graphique ont par ailleurs permis de valider concrètement ces résultats et de proposer un outil exploitable dans un cadre pratique par le département de mathématiques de l'Université Akli Mohand Oulhadj de Bouira.

Malgré ces résultats encourageants, certaines limites subsistent, notamment la dépendance à la qualité et à la disponibilité des données ainsi que la simplification de certaines contraintes du problème réel. Plusieurs pistes d'amélioration peuvent être envisagées :

— l'intégration de nouvelles contraintes, telles que les disponibilités hebdomadaires

des enseignants ou les contraintes de salles ;
- la prise en compte d'objectifs multiples (équilibrage des charges, satisfaction maximale, équité entre enseignants) au moyen d'une formulation multi-objectifs ;
- l'extension du modèle à plusieurs semestres simultanés, voire à l'ensemble de l'année universitaire ;
- l'utilisation de méthodes heuristiques ou méta-heuristiques (algorithmes génétiques, recuit simulé) pour traiter des instances de plus grande taille, par exemple à l'échelle de toute la faculté ;
- l'enrichissement de la fonction de score d'adéquation par une analyse plus fine des spécialités et des retours d'expérience des enseignants au fil des semestres.

En définitive, ce travail met en évidence l'efficacité des approches d'optimisation combinatoire pour améliorer le processus d'affectation des enseignants aux modules. Il fournit des résultats concrets et comparatifs permettant de guider le choix de la méthode la plus appropriée selon le contexte, et ouvre la voie à une automatisation plus poussée de la planification pédagogique au sein des établissements universitaires.

Bibliographie

[1] N. Belharrat, Y. Sadaoui et H. Ben Messaoud. La théorie des graphes : recherche opérationnelle (cours illustré et exercices corrigés). Alger : Pages Bleues, 2016.

[2] Claude Berge. « Two Theorems in Graph Theory ». In : Proceedings of the National Academy of Sciences of the USA 43.9 (1957), p. 842-844.

[3] Dimitris Bertsimas et John N. Tsitsiklis. Introduction to Linear Optimization. Belmont, MA : Athena Scientific, 1997.

[4] J. A. Bondy et U. S. R. Murty. Graph Theory with Applications. London : Macmillan, 1976.

[5] Rainer E. Burkard, Mauro Dell'Amico et Silvano Martello. Assignment Problems. Philadelphia, PA : SIAM, 2009.

[6] Thomas H. Cormen et al. Introduction to Algorithms. 3e éd. Cambridge, MA : MIT Press, 2009.

[7] Susanna S. Epp. Discrete Mathematics with Applications. 4e éd. Stamford, CT : Cengage Learning, 2011.

[8] Leonhard Euler. « Solutio problematis ad geometriam situs pertinentis ». In : Commentarii academiae scientiarum Petropolitanae 8 (1736), p. 128-140.

[9] L. R. Ford et D. R. Fulkerson. « Maximal Flow Through a Network ». In : Canadian Journal of Mathematics 8 (1956), p. 399-404.

[10] Jean-Claude Fournier. Théorie des graphes et applications. Paris : Lavoisier, 2011.

[11] GNU Project. GLPK (GNU Linear Programming Kit) – Reference Manual. 2024. url : https://www.gnu.org/software/glpk/ (visité le 15/12/2025).

[12] Aric A. Hagberg, Daniel A. Schult et Pieter J. Swart. « Exploring Network Structure, Dynamics, and Function Using NetworkX ». In : Proceedings of the 7th Python in Science Conference (SciPy 2008). Sous la dir. de Gaël Varoquaux, Travis Vaught et Jarrod Millman. Pasadena, CA, 2008, p. 11-15.

[13] Charles R. Harris et al. « Array Programming with NumPy ». In : Nature 585 (2020), p. 357-362.

[14] Frederick S. Hillier et Gerald J. Lieberman. Introduction to Operations Research. 10e éd. New York : McGraw-Hill, 2015.

[15] John E. Hopcroft et Richard M. Karp. « An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs ». In : SIAM Journal on Computing 2.4 (1973), p. 225-231.

[16] Dénes König. « Gráfok és mátrixok ». In : Matematikai és Fizikai Lapok 38 (1931), p. 116-119.

[17] Harold W. Kuhn. « The Hungarian Method for the Assignment Problem ». In : Naval Research Logistics Quarterly 2.1–2 (1955), p. 83-97.

[18] László Lovász et Michael D. Plummer. Matching Theory. Amsterdam : North-Holland, 1986.

[19] James Munkres. « Algorithms for the Assignment and Transportation Problems ». In : Journal of the Society for Industrial and Applied Mathematics 5.1 (1957), p. 32-38.

[20] George L. Nemhauser et Laurence A. Wolsey. Integer and Combinatorial Optimization. New York : Wiley, 1988.

[21] Christos H. Papadimitriou et Kenneth Steiglitz. Combinatorial Optimization : Algorithms and Complexity. Mineola, NY : Dover Publications, 1998.

[22] Python Software Foundation. Tkinter – Python Interface to Tcl/Tk. 2024. url : https://docs.python.org/3/library/tkinter.html (visité le 15/12/2025).

[23] Pauli Virtanen et al. « SciPy 1.0 : Fundamental Algorithms for Scientific Computing in Python ». In : Nature Methods 17 (2020), p. 261-272.

[24] Douglas B. West. Introduction to Graph Theory. 2e éd. Upper Saddle River, NJ : Prentice Hall, 2001.

[25] Wayne L. Winston. Operations Research : Applications and Algorithms. 4e éd. Belmont, CA : Cengage Learning, 2004.

[26] Laurence A. Wolsey. Integer Programming. New York : Wiley-Interscience, 1998.

Annexe A

Données utilisées

A.1 Données complètes des préférences

Cette annexe présente l'ensemble des préférences des enseignants utilisées dans notre étude.

TABLE A.1 – Table complète des préférences enseignants-modules

id_ens	**id_mod**	**pref**
E001	M001	5
E001	M002	6
E001	M003	7
E001	M010	7
E001	M013	9
E002	M001	7
E002	M002	6
E002	M003	9
E002	M010	8
E002	M013	6
E003	M001	7
E003	M002	6
E003	M003	10
E003	M004	7
E003	M010	10

id_ens	id_mod	pref
E003	M011	5
E003	M014	9
E004	M001	9
E004	M002	8
E004	M003	5
E004	M004	8
E004	M005	4
E004	M008	7
E004	M009	8
E004	M011	9
E005	M001	8
E005	M002	9
E005	M004	8
E005	M005	5
E005	M007	9
E005	M008	7
E005	M011	8
E006	M001	9
E006	M002	6
E006	M004	7
E006	M005	8
E006	M007	4
E006	M008	4
E006	M011	6
E006	M012	9
E006	M013	7
E007	M001	8
E007	M002	7

id_ens	**id_mod**	**pref**
E007	M004	10
E007	M005	8
E007	M007	6
E007	M008	4
E007	M011	7
E007	M012	8
E007	M013	8
E008	M001	8
E008	M002	6
E008	M004	7
E008	M005	8
E008	M007	5
E008	M008	8
E008	M009	9
E008	M011	9
E008	M012	8
E009	M001	8
E009	M002	5
E009	M003	5
E009	M005	8
E009	M006	7
E009	M010	5
E009	M012	9
E009	M013	9
E010	M001	8
E010	M002	4
E010	M003	9
E010	M004	5

id_ens	id_mod	pref
E010	M005	8
E010	M006	8
E010	M008	6
E010	M009	9
E010	M010	9
E010	M011	5
E010	M012	6
E010	M014	7

A.2 Scores d'adéquation enseignant-module

Ce tableau présente l'ensemble des scores d'adéquation enseignant-module calculés selon les critères du modèle.

TABLE A.2 – Table complète des scores globaux des affectations

id_ens	id_mod	s_{spec}	s_{grade}	s_{pref}	Score total
E001	M001	7	6	5	6.3
E001	M002	7	6	6	6.5
E001	M003	8	6	7	7.2
E001	M010	8	8	7	7.8
E001	M013	10	8	9	9.2
E002	M001	7	6	7	6.7
E002	M002	7	6	6	6.5
E002	M003	8	6	9	7.6
E002	M010	8	8	8	8.0
E002	M013	10	8	6	8.6
E003	M001	6	8	7	6.8
E003	M002	6	8	6	6.6
E003	M003	10	8	10	9.4

id_ens	**id_mod**	s_{spec}	s_{grade}	s_{pref}	**Score total**
E003	M004	6	9	7	7.1
E003	M010	10	9	10	9.7
E003	M011	6	9	5	6.7
E003	M014	7	9	9	8.0
E004	M001	10	6	9	8.6
E004	M002	7	6	8	6.9
E004	M003	6	6	5	5.8
E004	M004	10	8	8	9.0
E004	M005	5	8	4	5.7
E004	M008	9	8	7	8.3
E004	M009	9	8	8	8.5
E004	M011	9	8	9	8.7
E005	M001	10	6	8	8.4
E005	M002	7	6	9	7.1
E005	M004	10	8	8	9.0
E005	M005	5	8	5	5.9
E005	M007	7	8	9	7.7
E005	M008	9	8	7	8.3
E005	M011	9	8	8	8.5
E006	M001	8	8	9	8.2
E006	M002	6	8	6	6.6
E006	M004	8	10	7	8.4
E006	M005	8	10	8	8.6
E006	M007	6	10	4	6.8
E006	M008	6	10	4	6.8
E006	M011	7	10	6	7.7
E006	M012	9	10	9	9.3
E006	M013	9	10	7	8.9

id_ens	**id_mod**	s_{spec}	s_{grade}	s_{pref}	**Score total**
E007	M001	8	8	8	8.0
E007	M002	6	8	7	6.8
E007	M004	8	10	10	9.0
E007	M005	8	10	8	8.6
E007	M007	6	10	6	7.2
E007	M008	6	10	4	6.8
E007	M011	7	10	7	7.9
E007	M012	9	10	8	9.1
E007	M013	9	10	8	8.6
E008	M001	8	8	8	8.0
E008	M002	6	8	6	6.6
E008	M004	8	10	7	8.4
E008	M005	8	10	8	8.6
E008	M007	6	10	5	7.0
E008	M008	6	10	8	7.6
E008	M009	8	10	9	8.8
E008	M011	7	10	9	8.3
E008	M012	9	10	8	9.1
E009	M001	8	9	8	8.3
E009	M002	6	9	5	6.7
E009	M003	6	9	5	6.7
E009	M005	9	10	8	9.1
E009	M006	8	10	7	8.4
E009	M010	6	10	5	7.0
E009	M012	9	10	9	9.3
E009	M013	9	10	9	9.3
E010	M001	6	6	8	6.4
E010	M002	6	6	4	5.6

id_ens	**id_mod**	s_{spec}	s_{grade}	s_{pref}	**Score total**
E010	M003	9	6	9	8.1
E010	M004	6	8	5	6.4
E010	M005	8	8	8	8.0
E010	M006	7	8	8	7.5
E010	M008	6	8	6	6.6
E010	M009	8	8	9	8.2
E010	M010	9	8	9	8.7
E010	M011	7	8	5	6.9
E010	M012	6	8	6	6.6
E010	M014	8	8	7	7.8

Annexe B

Bibliothèques Python utilisées

Dans le cadre de l'implémentation du système d'affectation des enseignants aux modules, plusieurs bibliothèques Python ont été utilisées. Ces bibliothèques interviennent dans la manipulation des données, la résolution des problèmes d'optimisation, la modélisation des graphes ainsi que la visualisation des résultats.

B.1 NumPy

La bibliothèque `NumPy` est utilisée pour la manipulation efficace des tableaux et matrices numériques. Elle facilite les calculs matriciels et les opérations nécessaires à la construction des matrices de coût utilisées dans l'algorithme hongrois.

B.2 Pandas

La bibliothèque `Pandas` permet la lecture, l'organisation et le traitement des données stockées dans les fichiers CSV. Les données sont manipulées sous forme de `DataFrames`, ce qui simplifie leur exploitation dans les différentes étapes du projet.

B.3 SciPy

La bibliothèque `SciPy` est utilisée pour résoudre les problèmes d'optimisation combinatoire. La fonction :

```
scipy.optimize.linear_sum_assignment
```

permet d'obtenir une affectation optimale entre les enseignants et les modules à partir d'une matrice de coût.

B.4 Collections

Le module standard `collections`, notamment la structure `deque`, est utilisé pour gérer efficacement les files de traitement lors des parcours en largeur (BFS) dans l'algorithme de Hopcroft–Karp.

B.5 Tkinter

La bibliothèque `Tkinter` est utilisée pour développer l'interface graphique de l'application. Elle permet à l'utilisateur d'interagir facilement avec le système d'affectation.

B.6 Matplotlib

La bibliothèque `Matplotlib` permet la représentation graphique des résultats obtenus. Elle est utilisée pour produire des visualisations facilitant l'analyse des affectations générées par les algorithmes.

B.7 NetworkX

La bibliothèque `NetworkX` est utilisée pour la modélisation et la manipulation des graphes bipartis. Elle facilite la représentation des relations entre les enseignants et les modules ainsi que certaines opérations d'analyse sur les graphes.

Annexe C

Implémentation de l'algorithme de Hopcroft–Karp

Le code ci-dessous implémente l'algorithme de Hopcroft–Karp en deux étapes : une première phase calcule un couplage maximum classique, puis une seconde phase complète l'affectation pour les modules non encore attribués, en autorisant chaque enseignant à prendre en charge plusieurs modules dans la limite d'une capacité donnée.

```
import pandas as pd
from collections import deque


# ============================================================
# ALGORITHME DE HOPCROFT-KARP
# Calcule un couplage maximum dans un graphe biparti
# ============================================================
def hopcroft_karp(graph, U, V):
    """
    graph : dictionnaire d'adjacence {u : [v1, v2, ...]}
       representant le graphe biparti
    U     : liste des sommets de la classe gauche (enseignants)
    V     : liste des sommets de la classe droite (modules)
    Retourne le dictionnaire pairU = {u : v ou None}
       representant le couplage trouve.
    """

    # pairU[u] = module appariE a l'enseignant u (ou None s'il
       est libre)
    # pairV[v] = enseignant appariE au module v (ou None s'il
       est libre)
    pairU = {u: None for u in U}
```

```
    pairV = {v: None for v in V}

    # dist[u] = distance (niveau BFS) du sommet u depuis les
       sommets libres
    dist = {}
# ============================================================
    # Phase BFS : construit les niveaux et detecte une chaine
       augmentante
# ============================================================
    def bfs():
        queue = deque()

        # Initialisation : tous les sommets libres de U sont au
           niveau 0
        for u in U:
            if pairU[u] is None:
                dist[u] = 0
                queue.append(u)
            else:
                dist[u] = float('inf')

        found = False  # vrai si au moins une chaine
           augmentante existe

        # Parcours en largeur niveau par niveau
        while queue:
            u = queue.popleft()
            for v in graph[u]:

                # On ignore les sommets qui ne sont pas dans V
                if v not in pairV:
                    continue

                # Si le partenaire de v n'a pas encore ete
                   visite, on le marque
                if pairV[v] is not None and dist[pairV[v]] ==
                   float('inf'):
                    dist[pairV[v]] = dist[u] + 1
                    queue.append(pairV[v])

                # v est libre : on a trouve une chaine
```

```
                    augmentante
                if pairV[v] is None:
                    found = True

        return found

# ============================================================
    # Phase DFS : tente d'exploiter une chaine augmentante a
       partir de u
# ============================================================
    def dfs(u):
        for v in graph[u]:

            if v not in pairV:
                continue

            # On peut ameliorer le couplage si v est libre, ou
               si son
            # partenaire actuel peut etre lui-meme reapparie
            if pairV[v] is None or (
                dist[pairV[v]] == dist[u] + 1 and dfs(pairV[v])
            ):
                pairU[u] = v
                pairV[v] = u
                return True

        # Echec : on marque u comme inaccessible pour ne pas le
            re-explorer
        dist[u] = float('inf')
        return False

# ============================================================
    # Boucle principale : alterner BFS et DFS jusqu'a
       saturation
# ============================================================
    matching = 0
    while bfs():
        for u in U:
            if pairU[u] is None:
                if dfs(u):
                    matching += 1
```

```
    return pairU


# ============================================================
# AFFECTATION EN DEUX ETAPES
# Etape 1 : couplage maximum (1 module / enseignant)
# Etape 2 : redistribution des modules restants jusqu'a la
   capacite
# ============================================================
def affectation_hk_2_etapes(fichier_ens, fichier_mod,
   fichier_hab, capacite=2):
    """
    fichier_ens : CSV des enseignants
    fichier_mod : CSV des modules
    fichier_hab : CSV des habilitations (matrice 0/1)
    capacite    : nombre maximum de modules par enseignant
    """

    # Lecture des fichiers CSV
    df_ens = pd.read_csv(fichier_ens, sep=';')
    df_mod = pd.read_csv(fichier_mod, sep=';')
    df_hab = pd.read_csv(fichier_hab, sep=';')

    enseignants = df_ens.iloc[:, 0].tolist()
    modules = df_mod.iloc[:, 0].tolist()

    # Construction du graphe d'habilitation : un enseignant est
        relie
    # a un module si la cellule correspondante vaut 1
    graph = {e: [] for e in enseignants}
    for _, row in df_hab.iterrows():
        e = row.iloc[0]
        for m in modules:
            if m in row.index and str(row[m]) == '1':
                graph[e].append(m)
# ============================================================
    # ETAPE 1 : couplage maximum classique
# ============================================================
    print("\n===== ETAPE 1 =====\n")
    pairU = hopcroft_karp(graph, enseignants, modules)
```

```
    affectation = []
    modules_restants = set(modules)
    utilisation = {e: 0 for e in enseignants}

    for e, m in pairU.items():
        if m:
            affectation.append((e, m))
            modules_restants.discard(m)
            utilisation[e] += 1
            print(f"{e} ---> {m}")

# ============================================================
    # ETAPE 2 : on duplique virtuellement les enseignants pour
       permettre
    # plusieurs modules par enseignant, dans la limite de la
       capacite
# ============================================================
    print("\n===== ETAPE 2 =====\n")

    if modules_restants:
        modules_restants = list(modules_restants)

        # Pour chaque enseignant, on cree autant de "copies"
           que de places
        # restantes dans sa capacite
        U_expanded = []
        for e in enseignants:
            reste = capacite - utilisation[e]
            for i in range(reste):
                U_expanded.append(f"{e}_{i}")

        # Chaque copie herite des memes habilitations que l'
           original
        graph2 = {u: [] for u in U_expanded}
        for u in U_expanded:
            e = u.split('_')[0]
            graph2[u] = [m for m in graph[e] if m in
               modules_restants]

        # Nouveau couplage maximum sur les copies vs modules
```

```
            restants
        pairU2 = hopcroft_karp(graph2, U_expanded,
            modules_restants)

        for u, m in pairU2.items():
            if m:
                e = u.split('_')[0]
                affectation.append((e, m))
                print(f"{e} ---> {m}")

    print("\n===== RESULTAT FINAL =====")
    print(len(affectation), "affectations")
    return affectation


# ============================================================
# Point d'entree du programme
# ============================================================
if __name__ == "__main__":
    affectation_hk_2_etapes(
        "enseignants01.csv",
        "modules01.csv",
        "habilitation01.csv",
        capacite=2
    )
```

Annexe D

Implémentation de l'algorithme Hongrois

Le code suivant met en œuvre l'algorithme Hongrois (fonction `linear_sum_assignment` de SciPy) sur le problème d'affectation pondéré. Comme la fonction de SciPy minimise par défaut le coût total, la matrice de scores est niée ($C_{\text{neg}} = -C$) afin de se ramener à un problème de minimisation équivalent.

```
import numpy as np
import pandas as pd
from scipy.optimize import linear_sum_assignment


# ============================================================
# AFFECTATION EN DEUX ETAPES PAR L'ALGORITHME HONGROIS
# Etape 1 : affectation 1-1 optimale via linear_sum_assignment
# Etape 2 : redistribution des modules non affectes en
   respectant
#           la capacite maximale de chaque enseignant
# ============================================================
def affectation_2_hungarian(fichier_ens, fichier_mod,
   fichier_score, capacite=3):
    """
    fichier_ens   : CSV des enseignants
    fichier_mod   : CSV des modules
    fichier_score : CSV des scores d'adequation
    capacite      : nombre maximum de modules par enseignant
    """

    # Lecture des donnees
    df_ens = pd.read_csv(fichier_ens, sep=';')
```

```
    df_mod = pd.read_csv(fichier_mod, sep=';')
    df_score = pd.read_csv(fichier_score, sep=';')

    enseignants = df_ens.iloc[:, 0].tolist()
    modules = df_mod.iloc[:, 0].tolist()

    affectation = []
    score_total = 0

    n = len(enseignants)
    m = len(modules)

    # On rend la matrice carree en ajoutant des lignes/colonnes
        "DUMMY"
    # car linear_sum_assignment exige une matrice rectangulaire
        bien definie
    size = max(n, m)
    enseignants_ext = enseignants + [f"DUMMY_{i}" for i in
       range(size - n)]
    modules_ext = modules + [f"DUMMY_M{i}" for i in range(size
       - m)]

# ============================================================
    # Construction de la matrice de couts C[i,j]
# ============================================================
    C = np.zeros((size, size))
    for i, ens in enumerate(enseignants_ext):
        for j, mod in enumerate(modules_ext):

            # Les paires impliquant un sommet "DUMMY" recoivent
                un score nul
            if "DUMMY" in ens or "DUMMY" in mod:
                C[i][j] = 0
            else:
                # Recherche du score correspondant dans le
                   fichier
                ligne = df_score[
                    (df_score.iloc[:, 0] == ens) &
                    (df_score.iloc[:, 1] == mod)
                ]
                if not ligne.empty:
```

```
                    C[i][j] = ligne.iloc[0, 2]

    # On nie la matrice pour transformer le probleme de
       maximisation
    # en probleme de minimisation, conformement a la convention
        SciPy
    C_neg = -C
    lignes, colonnes = linear_sum_assignment(C_neg)

# ============================================================
    # ETAPE 1 : extraction des affectations reelles (sans DUMMY
       )
# ============================================================
    modules_restants = set(modules)
    utilisation = {e: 0 for e in enseignants}

    for i, j in zip(lignes, colonnes):
        ens = enseignants_ext[i]
        mod = modules_ext[j]

        if "DUMMY" not in ens and "DUMMY" not in mod:
            score = C[i][j]
            affectation.append((ens, mod, score))
            modules_restants.discard(mod)
            utilisation[ens] += 1
            score_total += score
            print(f"{ens} ---> {mod} | score = {score}")

# ============================================================
    # ETAPE 2 : affectation des modules restants en duplicant
       les
    # enseignants n'ayant pas encore atteint leur capacite
       maximale
# ============================================================
    if modules_restants:
        modules_restants = list(modules_restants)

        # Liste des "places" encore disponibles pour chaque
           enseignant
        ens_expanded = []
        for e in enseignants:
```

```
            reste = capacite - utilisation[e]
            for _ in range(reste):
                ens_expanded.append(e)

        n2 = len(ens_expanded)
        m2 = len(modules_restants)

        if n2 > 0 and m2 > 0:
            # Nouvelle matrice de couts pour la phase 2
            C2 = np.zeros((n2, m2))
            for i, ens in enumerate(ens_expanded):
                for j, mod in enumerate(modules_restants):
                    ligne = df_score[
                        (df_score.iloc[:, 0] == ens) &
                        (df_score.iloc[:, 1] == mod)
                    ]
                    if not ligne.empty:
                        C2[i][j] = ligne.iloc[0, 2]

            # Penalite forte sur les couples non habilites pour
                eviter
            # qu'ils ne soient choisis par defaut
            C2_neg = -C2
            C2_neg[C2 == 0] = 1e6

            lignes2, colonnes2 = linear_sum_assignment(C2_neg)

            for i, j in zip(lignes2, colonnes2):
                ens = ens_expanded[i]
                mod = modules_restants[j]
                score = C2[i][j]

                # On ne valide l'affectation que si le score
                   est strictement positif
                if score > 0:
                    affectation.append((ens, mod, score))
                    score_total += score
                    print(f"{ens} ---> {mod} | score = {score}"
                       )

    print("\n===== RESULTAT FINAL =====")
```

```
    print("Nombre d'affectations :", len(affectation))
    print("Score total :", score_total)
    return affectation, score_total


# ================================================================
# Point d'entree du programme
# ================================================================
if __name__ == "__main__":
    affectation_2_hungarian(
        "enseignants01.csv",
        "modules01.csv",
        "score1.csv",
        capacite=3
    )
```

Annexe E

Modélisation du problème d'affectation sous GLPK (PLNE)

Le modèle GLPK ci-dessous traduit en programmation linéaire en nombres entiers le problème étudié au chapitre 3. Il maximise le score total d'adéquation sous l'ensemble des contraintes (couverture des modules, charge horaire maximale, habilitations).

```
# ============================================================
# Modele GLPK / GMPL pour l'affectation enseignants-modules
# Objectif : maximiser le score total d'adequation
# ============================================================

# ---- Ensembles ----
set I;          # ensemble des enseignants
set J;          # ensemble des modules

# ---- Parametres ----
param C{I,J} default 0;     # score d'adequation entre
   enseignant i et module j
param h{I,J} default 0;     # indicateur d'habilitation (0 ou
   1)
param VH{J};                # volume horaire de chaque module
param CHmax{I};             # charge horaire maximale de chaque
    enseignant

# ---- Variables de decision ----
var x{I,J} binary;          # x[i,j] = 1 si l'enseignant i est
   affecte au module j

# ---- Fonction objectif : maximiser le score total ----
maximize Total_Score:
```

```
    sum {i in I, j in J} C[i,j] * x[i,j];

# ---- Contraintes ----

# (C2) Chaque module est affecte a exactement un enseignant
s.t. Un_Prof_Par_Module {j in J}:
    sum {i in I} x[i,j] = 1;

# Chaque enseignant prend en charge au moins un module
s.t. Chaque_Prof_Au_Moins_Un_Module {i in I}:
    sum {j in J} x[i,j] >= 1;

# (C3) Respect des habilitations
s.t. Verif_Habilitation {i in I, j in J}:
    x[i,j] <= h[i,j];

# (C4) Respect de la charge horaire maximale
s.t. Charge_Horaire {i in I}:
    sum {j in J} VH[j] * x[i,j] <= CHmax[i];

solve;

display x;
display Total_Score;

# ============================================================
# DONNEES NUMERIQUES
# ============================================================
data;

set I := E001 E002 E003 E004 E005 E006 E007 E008 E009 E010;

set J := M001 M002 M003 M004 M005 M006 M007 M008 M009
         M010 M011 M012 M013 M014;

# Volumes horaires des modules (en heures)
param VH := M001 6   M002 3   M003 3   M004 6   M005 4.5
            M006 3   M007 3   M008 6   M009 3   M010 3
            M011 4.5 M012 4.5 M013 4.5 M014 6;

# Charges horaires maximales des enseignants
```

```
param CHmax := E001 9    E002 9    E003 4.5  E004 9    E005 9
               E006 6    E007 9    E008 9    E009 9    E010 9;

# Matrice des scores d'adequation
param C :=
[E001,*] M001 6.3 M002 6.5 M003 7.2 M010 7.8 M013 9.2
[E002,*] M001 6.7 M002 6.5 M003 7.6 M010 8.0 M013 8.6
[E003,*] M001 6.8 M002 6.6 M003 9.4 M004 7.1 M010 9.7 M011 6.7
   M014 8.0
[E004,*] M001 8.6 M002 6.9 M003 5.8 M004 9.0 M005 5.7 M008 8.3
   M009 8.5 M011 8.7
[E005,*] M001 8.4 M002 7.1 M004 9.0 M005 5.9 M007 7.7 M008 8.3
   M011 8.5
[E006,*] M001 8.2 M002 6.6 M004 8.4 M005 8.6 M007 6.8 M008 6.8
   M011 7.7 M012 9.3 M013 8.9
[E007,*] M001 8.0 M002 6.8 M004 9.0 M005 8.6 M007 7.2 M008 6.8
   M011 7.9 M012 9.1 M013 8.6
[E008,*] M001 8.0 M002 6.6 M004 8.4 M005 8.6 M007 7.0 M008 7.6
   M009 8.8 M011 8.3 M012 9.1
[E009,*] M001 8.3 M002 6.7 M003 6.7 M005 9.1 M006 8.4 M010 7.0
   M012 9.3 M013 9.3
[E010,*] M001 6.4 M002 5.6 M003 8.1 M004 6.4 M005 8.0 M006 7.5
   M008 6.6 M009 8.2 M010 8.7 M011 6.9 M012 6.6 M014 7.8;

# Matrice des habilitations (1 = habilite, 0 = non habilite)
param h :=
[E001,*] M001 1 M002 1 M003 1 M010 1 M013 1
[E002,*] M001 1 M002 1 M003 1 M005 1 M010 1 M013 1
[E003,*] M001 1 M002 1 M003 1 M004 1 M010 1 M011 1 M014 1
[E004,*] M001 1 M002 1 M003 1 M004 1 M005 1 M008 1 M009 1 M011
   1
[E005,*] M001 1 M002 1 M004 1 M005 1 M007 1 M008 1 M011 1
[E006,*] M001 1 M002 1 M004 1 M005 1 M007 1 M008 1 M011 1 M012
   1 M013 1
[E007,*] M001 1 M002 1 M004 1 M005 1 M007 1 M008 1 M011 1 M012
   1 M013 1
[E008,*] M001 1 M002 1 M004 1 M005 1 M007 1 M008 1 M009 1 M011
   1 M012 1
[E009,*] M001 1 M002 1 M003 1 M005 1 M006 1 M010 1 M012 1 M013
   1
[E010,*] M001 1 M002 1 M003 1 M004 1 M005 1 M006 1 M008 1 M009
```

```
   1 M010 1 M011 1 M012 1 M014 1;

end;
```

Annexe F

Interface utilisateur du système d'affectation

L'interface graphique a été développée avec la bibliothèque `Tkinter`. Elle permet à l'utilisateur de charger les fichiers CSV des enseignants, des modules et des scores, de choisir l'algorithme d'affectation (Hongrois ou Hopcroft–Karp), de définir la capacité maximale par enseignant, puis de lancer le calcul et de visualiser les résultats.

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from scipy.optimize import linear_sum_assignment
from collections import deque


# ============================================================
# CLASSE PRINCIPALE DE L'APPLICATION GRAPHIQUE
# ============================================================
class AcademicAssignmentApp:
    def __init__(self, root):
        # Initialisation de la fenetre principale
        self.root = root
        self.root.title("Systeme d'Affectation Master - Version
            Finale Complete")
        self.root.geometry("1100x800")

        # Dictionnaire des chemins vers les fichiers CSV
           charges
```

```
        self.files = {"ens": None, "mod": None, "score": None}

        # Liste des affectations finales (couples enseignant-
            module)
        self.final_matches = []

        # Construction de l'interface
        self.setup_ui()

# ============================================================
    # Construction de l'interface graphique
# ============================================================
    def setup_ui(self):
        # Bandeau d'en-tete
        header = tk.Frame(self.root, bg="#2c3e50", pady=15)
        header.pack(fill="x")
        tk.Label(
            header,
            text="AFFECTATION LOGIQUE : HUNGARIAN & HOPCROFT",
            fg="white", font=("Arial", 14, "bold"), bg="#2c3e50
                "
        ).pack()

        # Cadre pour le chargement des fichiers
        file_frame = tk.LabelFrame(
            self.root, text="Chargement des donnees", padx=15,
                pady=10
        )
        file_frame.pack(fill="x", padx=20, pady=10)

        tk.Button(file_frame, text="Enseignants",
                  command=lambda: self.set_file("ens")).grid(
                      row=0, column=0)
        tk.Button(file_frame, text="Modules",
                  command=lambda: self.set_file("mod")).grid(
                      row=0, column=1)
        tk.Button(file_frame, text="Scores",
                  command=lambda: self.set_file("score")).grid(
                      row=0, column=2)

        # Cadre pour les parametres de calcul
```

```
        ctrl = tk.LabelFrame(self.root, text="Parametres", padx
            =15, pady=10)
        ctrl.pack(fill="x", padx=20, pady=10)

        # Choix de l'algorithme
        self.algo_box = ttk.Combobox(
            ctrl,
            values=["Hungarian (Pondere)", "Hopcroft-Karp (
                Binaire)"],
            state="readonly", width=25
        )
        self.algo_box.current(0)
        self.algo_box.grid(row=0, column=1)

        # Saisie de la capacite maximale par enseignant
        self.cap_spin = tk.Spinbox(ctrl, from_=1, to=10, width
            =5)
        self.cap_spin.grid(row=0, column=3)

        # Bouton de lancement
        tk.Button(ctrl, text="Calculer", bg="#27ae60", fg="
            white",
                  command=self.process_assignment).grid(row=0,
                      column=4)

        # Tableau d'affichage des resultats
        self.tree = ttk.Treeview(
            self.root,
            columns=("M", "E", "S"), show="headings"
        )
        self.tree.heading("M", text="Module")
        self.tree.heading("E", text="Enseignant")
        self.tree.heading("S", text="Score / Match")
        self.tree.pack(expand=True, fill="both")

# ============================================================
    # Selection d'un fichier CSV via une boite de dialogue
# ============================================================
    def set_file(self, key):
        path = filedialog.askopenfilename(filetypes=[("CSV", "
            *.csv")])
```

```
        if path:
            self.files[key] = path

# ============================================================
    # Implementation simplifiee de l'algorithme de Hopcroft-
       Karp
    # (utilisee en mode "Binaire" depuis l'interface)
# ============================================================
    def run_hopcroft_karp(self, graph, U, V):
        pairU = {u: None for u in U}
        pairV = {v: None for v in V}
        dist = {}

        def bfs():
            queue = deque()
            for u in U:
                if pairU[u] is None:
                    dist[u] = 0
                    queue.append(u)
                else:
                    dist[u] = float('inf')
            found = False
            while queue:
                u = queue.popleft()
                for v in graph.get(u, []):
                    if v in pairV and pairV[v] is None:
                        found = True
            return found

        def dfs(u):
            for v in graph.get(u, []):
                if v in pairV:
                    pairU[u] = v
                    pairV[v] = u
                    return True
            return False

        while bfs():
            for u in U:
                if pairU[u] is None:
                    dfs(u)
```

```
        return pairU

# ================================================================
    # Calcul de l'affectation et mise a jour de l'interface
# ================================================================
    def process_assignment(self):
        # A completer : appel de l'algorithme choisi, mise a
           jour du tableau
        pass

# ================================================================
    # Visualisation graphique du couplage obtenu
# ================================================================
    def draw_graph(self):
        # A completer : utilisation de NetworkX et Matplotlib
        pass
# ================================================================
    # Sauvegarde des resultats au format CSV
# ================================================================
    def save_to_csv(self):
        # A completer : export du tableau d'affectation
        pass


# ================================================================
# Point d'entree du programme
# ================================================================
if __name__ == "__main__":
    root = tk.Tk()
    app = AcademicAssignmentApp(root)
    root.mainloop()
```