

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Formations par Alternance / Informatique industrielle par apprentissage

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 11 26

polytech.univ-tours.fr

Rapport de projet développement DII3

2017-2018

Outil d'aide à l'affectation des projets

Tuteur académique

Ameur SOUKHAL

Étudiants

Jérémy LOCHE (DII3)

Louis THOMAS (DII3)

24 avril 2018



Liste des intervenants

Nom	Email	Qualité
Jérémy LOCHE	jeremy.loche@etu.univ-tours.fr	Étudiant DII3
Louis THOMAS	louis.thomas@etu.univ-tours.fr	Étudiant DII3
Ameur SOUKHAL	ameur.soukhal@univ-tours.fr	Tuteur académique, Informatique industrielle par apprentissage



Avertissement

Ce document a été rédigé par Jérémy Loche et Louis Thomas susnommés les auteurs.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Ameer Soukhal susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Jérémy Loche et Louis Thomas, *Outil d'aide à l'affectation des projets*, Rapport de projet développement DII3, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={Loche, Jérémy and Thomas, Louis},
  title={Outil d'aide à l'affectation des projets},
  type={Rapport de projet développement DII3},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iii
Liste des tableaux	iv
Liste des Algorithmes	v
Introduction	1
1 Le problème d'affectation	2
1 Présentation	2
2 La modélisation mathématique.....	2
2.1 Les données et les paramètres	2
2.2 Les variables et les contraintes	3
2.3 Objectif simple, maximiser la préférence moyenne.....	4
2.4 Objectif de compromis, la contrainte de la préférence minimale	5
2 Organisation et choix techniques de réalisation de l'outils d'affectation	8
1 Résoudre le problème d'affectation	8
1.1 L'algorithme Hongrois, de Kuhn-Munkres	8
1.2 Résolution par un Solver Linéaire.....	10

2	Implémentation de la méthode de résolution.....	11
3	Interface de saisie des données et création d'un back-end	12
3.1	Les choix et les objectifs	12
3.2	Réalisation de l'interface de saisie	13
3.3	Les améliorations	15
Conclusion		17
Annexes		18
4	Code GMPL pour le modèle mathématique d'affectation	18



Table des figures

2 Organisation et choix techniques de réalisation de l'outils d'affectation

1	Modélisation étudiants-projets dans un graphe biparti complet.....	9
2	Couplages parfait sur le graphe biparti (Figure 1 (Chapitre 2))	9
3	Architecture de l'application de résolution des affectations.....	13
4	Interface globale de l'application.....	14
5	Tableau de saisie des préférences.....	14
6	Saisies des paramètres d'affectation.....	14
7	Bouton affecter	14



Liste des tableaux

1 Le problème d'affectation

1	Indiçage des entités du modèle mathématique	3
2	Données de préférence pour l'exemple.....	5



Liste des Algorithmes

1	Résolution du modèle avec <i>ValeurPrefMin</i> choisi par tentative	6
2	Recherche de <i>ValeurPrefMin</i> dichotomique	7



Introduction

Tous les ans, apparaissent dans les promotions d'étudiants ingénieurs de Polytech Tours, d'intenses débats au moment des affectations de projets. En effet, chacun souhaite avoir le projet qui lui plaît le plus. Pour aider à dénouer ces débats, nous avons proposé de travailler sur un outil d'aide à l'affectation des projets étudiants.

Quels dispositifs peuvent être mis en place pour aider à dénouer ces problèmes ?

Vous découvrirez dans un premier temps comment nous avons modélisé ce problème à travers l'optimisation linéaire. L'idée qui sera développée ici est comment utiliser les votes des étudiants pour leur projet préféré pour trouver l'affectation qui conviendra à tout le monde.

Dans un second temps, nous vous présenterons comment se passe la phase d'optimisation à travers le solveur GLPK. A ce stade on abordera les méthodes de résolution via l'algorithme Hongrois et par l'intermédiaire d'un solveur linéaire.

Nous terminerons par l'outil de saisie en ligne basé sur Google sheet que nous avons mis en place pour assister les gérants.

1

Le problème d'affectation

1 Présentation

Ce travail est un problème d'affectation. En effet, le but est d'associer à un ou plusieurs individus un projet précis en respectant un objectif simple : maximiser la satisfaction générale de l'affectation. L'idée est que chacun soit satisfait du projet auquel il a été affecté et qu'il soit en mesure de le mener à bien dans les meilleures conditions.

2 La modélisation mathématique

Pour résoudre ce problème, nous allons avoir besoin de le formaliser de manière mathématique afin d'essayer de trouver des solutions. Nous rappelons que le but est d'affecter un étudiant, un binôme ou un groupe de personnes à un projet.

Le fait de formaliser un problème en langage mathématique nous permettra de demander à un **solver** de trouver la solution à notre problème.

Pour cela, il faut définir quels sont les **paramètres**¹, les **variables**², les **contraintes**³, et la **fonction objectif**⁴ de notre problème.

Commençons par définir nos données c'est à dire les paramètres de notre modèle.

2.1 Les données et les paramètres

Pour cela on commence par créer deux entités appartenant à deux ensembles qui seront nos données de départ :

1. une entité *individu* \in *Individus* : représente une personne, un binôme ou un groupe de personnes qui accomplira un **projet** ;
2. une entité *projet* \in *Projets* : représente un projet auquel sera affecté un **individu**.

Pour plus de simplicité, on va indiquer les entités comme dans **Table 1**. On a choisi que les binômes seront représentés par des individus.

-
1. Dans un modèle mathématique, un paramètre est une donnée.
 2. Dans un modèle mathématique, une variable est une valeur calculée, une inconnue.
 3. Les contraintes représentent les règles à respecter lors de la résolution du problème.
 4. C'est l'ultime but à atteindre pour établir que la solution au problème est optimale.

Table 1 – Indichage des entités du modèle mathématique

$\forall \text{individu} \in \text{Individus}$		$\forall \text{projet} \in \text{Projets}$	
Indice Individu	Nom individu	Indice Projet	Nom projet
1	Binôme : Arthur, Léo	1	Projet : Lampe connecté
2	Binôme : Sophie, Jean	2	Projet : Moniteur UVA
...
nbIndividus	Binôme : Paul, Pierre	nbProjets	Projet : Voiture RC

Pour que l'affectation de chaque individu à un projet, il faut assez de projet pour tout le monde ce qui implique la relation suivante :

$$nbProjets \geq nbIndividus$$

Pour tout les individus, on souhaite connaître sa préférence pour un projet afin de procéder à l'affectation. En utilisant le formalisme mathématique, on peut définir la quantité *preference* qui définira l'appréciation d'un individu à un projet donné :

$$\forall (i, p) \in \text{Individus} \times \text{Projets}, preference[i, p] \in \llbracket 1, nbProjets \rrbracket$$

On choisit de dire que cette préférence va de *nbProjet* à 1 dans l'ordre décroissant d'appréciation. Ainsi, un individu qui aurait la préférence *nbProjets* pour un projet indiquera que ce projet est son préféré et 1 pour celui qui lui plait le moins.

Pour que l'affectation soit équitable, un individu ne peut pas donner la même préférence à deux projets différents. De manière mathématique cela donne :

$$\forall (i, p1, p2) \in \text{Individus} \times \text{Projets}^2, p1 \neq p2 \Rightarrow preference[i, p1] \neq preference[i, p2]$$

Lorsque ces pré-conditions sont respectés, alors on crée un **paramètre** appelé préférence utile pour résoudre le problème d'affectation.

$$\forall (i, p) \in \text{Individus} \times \text{Projets}, preference[i, p]$$

Nous allons maintenant voir quels contraintes et variables sont à déterminer pour résoudre le problème.

2.2 Les variables et les contraintes

Pour modéliser entièrement le problème d'affectation, la préférence de chaque individus pour un projet n'est pas suffisante. Nous allons devoir mettre en place des **variables** et des **contraintes** qu'elles doivent respecter.

Une **variable** est une valeur qui doit être déterminée pour résoudre le problème. Il faut donc la calculer.

Nous cherchons à déterminer a quel projet est affecté un individu. Pour cela on crée la variable *affectation* :

$$\forall (i, p) \in \text{Individu} \times \text{Projets}, affectation[i, p] = \begin{cases} 1 & \text{si l'individu } i \text{ est affecté au projet } p \\ 0 & \text{sinon} \end{cases}$$

On connaît la quantité que l'on souhaite déterminer pour résoudre le problème d'affectation, cependant, il faut définir des **contraintes** qui vont définir les règles de l'affectation.

Nous allons énoncer et définir deux contraintes utiles à la résolution du problème. La première est le nombre de projets par individus et la seconde est le nombre d'individus par projets. Pour une affectation d'un individu pour un projet, alors on a les contraintes (1) et (2).

$$\forall i \in \text{Individus}, nbProjetsParIndividu = \sum_{p \in Projets} affectation[i, p] = 1 \quad (1)$$

Pour notre problème, un individu doit être affecté à un seul projet, ce qui donne la contrainte (1).

$$\forall p \in Projets, nbIndividusParProjet = \sum_{i \in Individus} affectation[i, p] \leq 1 \quad (2)$$

La contrainte (2) dit qu'un projet ne peut être réalisé que par au plus un individu.

Nous avons défini nos paramètres, variables et contraintes, donc il ne reste plus qu'à définir l'objectif à atteindre avec tout ça.

2.3 Objectif simple, maximiser la préférence moyenne

Pour choisir quelle configuration de variable on va garder, c'est à dire quel choix d'affectation on va faire, on va devoir mettre en place un indicateur permettant de juger de la qualité de l'affectation. On appelle cet indicateur la **fonction objectif**. Trouver la solution à un problème mathématique se résume souvent à établir la bonne fonction objectif qu'on souhaite faire converger vers une valeur donnée, maximiser ou minimiser.

Dans notre cas, on souhaite maximiser la satisfaction générale, c'est à dire maximiser la somme des préférences des projets affectés à chacun. Ainsi le résultat optimal sera atteint lorsqu'on aura réussi à trouver la combinaison de variable qui donne la plus grande valeur de la **fonction objectif**.

$$\text{maximiser} \left(\sum_{(i \in Individus)} \sum_{(p \in Projets)} affectation[i, p] \times preference[i, p] \right) \quad (3)$$

Pour le problème d'affectation maximiser la fonction objectif (3) donnera la satisfaction générale la plus élevée car elle correspondra à faire en sorte que si un individu est affecté à un projet, la valeur de la préférence sera en moyenne la plus grande.

Cette méthode fonctionne est efficace pour trouver la meilleure préférence moyenne, cependant, elle peut être injuste et sélectionner pour la majorité des individus la préférence maximale et pour un individu la pire des préférences.

Il peut donc apparaître un sentiment d'injustice derrière cette affectation avec la sensation que quelqu'un a été mis de côté pour satisfaire le bien commun.

A cela, nous proposons une solution de compromis qui fournit une solution plus homogène (si elle existe évidemment).

2.4 Objectif de compromis, la contrainte de la préférence minimale

Pour obtenir une affectation plus homogène, on introduit une contrainte au modèle : c'est la contrainte de la préférence minimale. Le but est d'assurer que lorsqu'un individu est affecté à un projet, la préférence soit supérieure ou égale à une valeur de seuil *ValeurPrefMin*. On peut donc assurer avec cette contrainte que la préférence n'atteindra jamais 1 ou 2 etc...

On pose donc un nouveau **paramètre** :

$$ValeurPrefMin \in \llbracket 1, nbProjet \rrbracket$$

Et une nouvelle **contrainte**, qui impose que l'affectation soit faite avec une préférence supérieure à *ValeurPrefMin* :

$$\forall i \in Individus, preferenceMinimale = \sum_{p \in Projets} affectation[i, p] \times preference[i, p] \geqslant ValeurPrefMin$$

Problème

Cette contrainte est très restrictive sur les solutions. Il se peut même *ValeurPrefMin* de manière trop audacieuse et que le problème soit insoluble avec ces paramètres. Prenons un exemple de l'affectation de 4 individus et 4 projets (Cf. [Table 2](#)).

Table 2 – Données de préférence pour l'exemple

préférence[individu,projet]	Projet 1	Projet 2	Projet 3	Projet 4
Individu 1	4	3	2	1
Individu 2	4	3	1	2
Individu 3	3	4	2	1
Individu 4	2	4	1	3

Si on choisit *ValeurPrefMin* = 3, alors le modèle mathématique est insoluble car la contrainte *preferenceMinimale* ne pourra pas être respectée pour tout les individus. On a 4 projets pour 4 individus, on remarque qu'il faudra donc exactement 1 projet par individu pour respecter la contrainte *nbProjetsParIndividu* = 1. Le problème est que si un individu est affecté au projet P3, alors la valeur de la préférence est inférieure ou égale à 2 donc inférieure à *ValeurPrefMin* = 3. On ne respecte alors pas la contrainte *preferenceMinimale* pour un des individu donc le modèle est insoluble. Pour trouver une solution il faudra revoir le paramètre *ValeurPrefMin* à la baisse. Pour la valeur 2, on commencera à trouver des solutions pour laquelle la *ValeurPrefMin* est maximale. Donc les bonnes solutions de compromis.

Il y a plusieurs solutions pour automatiser la détermination de la valeur de *ValeurPrefMin* pour laquelle on assure que tout le monde aura la meilleur préférence possible tout en maximisant la satisfaction générale.

Solution 1

Le but est d'essayer de trouver la valeur maximale de *ValeurPrefMin* pour laquelle on arrive à résoudre le modèle. Pour cela, on peut rechercher la valeur maximale en faisant plusieurs résolutions du modèle mathématique en fixant différentes valeurs de *ValeurPrefMin*. On peut faire une recherche par tentative ou dichotomique. *ValeurPrefMin* n'intervient pas dans la fonction objectif à ce stade.

Recherche de ValeurPrefMin par tentative

La fonction *SolveModel*(param : *ValeurPrefMin*) est un sous-algorithme appelé par notre algorithme pour effectuer la résolution du modèle mathématique avec la valeur de préférence minimale passée en paramètre. Elle renvoie une structure *ResultSolver* contenant les résultats de la résolution *ResultSolver.data* et l'état de la résolution *ResultSolver.state* qui peut valoir soit *Success* si une solution au modèle a été trouvée et *Fail* sinon.

On l'utilise donc pour savoir si on a réussi à résoudre le

```

/* Init variable résultat du solver, contient ResultSolver.data et
   ResultSolver.state */
ResultSolver ← Null
/* Commencer par la valeur la plus restrictive jusqu'à la moins
   restrictive */
for ValeurPrefMin = nbProjets to 1 do
  /* Effectuer une résolution avec la valeur de préférence minimale */
  ResultSolver ← SolveModel(ValeurPrefMin)
  if ResultSolver.state = Success then
    /* Le solver a trouvé une solution */
    /* on a trouvé le maximum pour ValeurPrefMin */
    /* on arrête la boucle ici */
    break
  end
end
/* Retourner le résultat du solver qui est donc la solution avec la
   ValeurPrefMin maximale et la plus grande satisfaction */
return ResultSolver

```

Algorithme 1 : Résolution du modèle avec *ValeurPrefMin* choisi par tentative

Cette méthode n'a pas la meilleure complexité et peut considérablement allonger les temps de résolution du modèle. C'est pour cela que nous proposons la méthode dichotomique.

Recherche de ValeurPrefMin dichotomique

Solution 2

On change la nature de *ValeurPrefMin* et elle devient donc une **variable** intervenant dans la fonction objectif. L'idée derrière cette modification est de demander, en plus de la maximisation de la préférence moyenne, une maximisation de la *ValeurPrefMin*.

Méthode par aggrégation

On peut rajouter *ValeurPrefMin* dans la fonction objectif en faisant une simple somme (Cf. (4))

$$\text{maximiser} \left(\text{ValeurPrefMin} + \sum_{(i \in \text{Individus})} \sum_{(p \in \text{Projets})} \text{affectation}[i, p] \times \text{preference}[i, p] \right) \quad (4)$$

Cette agrégation permettra donc de chercher aussi à maximiser la *ValeurPrefMin*. Cependant, elle ne le sera pas au même titre que la satisfaction moyenne car il peut être plus avantageux de chercher à maximiser d'avantage la satisfaction que la *ValeurPrefMin*.

```

/* Init variable résultat du solver, contient ResultSolver.data et
   ResultSolver.state                                     */
ResultSolverFinal ← Null
deb ← 1
fin ← nbProjet
mil ←  $\frac{deb+fin}{2}$ 
while deb < fin do
  ResultSolverTemp = SolverModel(mil)
  if ResultSolverTemp.state = Success then
    deb = mil + 1
    ResultSolverFinal = ResultSolverTemp
    mil ←  $\frac{deb+fin}{2}$ 
  end
  else
    fin = mil - 1
    mil ←  $\frac{deb+fin}{2}$ 
  end
end
return ResultSolverFinal

```

Algorithme 2 : Recherche de *ValeurPrefMin* dichotomique**Méthode de combinaison linéaire des variables**

Pour cela, on peut ajouter des poids α, β deux nombres donnés qui peuvent servir à donner plus d'importance à la *ValeurPrefMin* ou à la satisfaction générale (Cf. (5)).

$$\text{maximiser} \left(\alpha \text{ValeurPrefMin} + \beta \sum_{(i \in \text{Individus})} \sum_{(p \in \text{Projets})} \text{affectation}[i, p] \times \text{preference}[i, p] \right) \quad (5)$$

Transition :

Après avoir vu le fond du problème que nous avons souhaité résoudre dans ce projet, nous allons passer à notre proposition de réalisation d'un outil d'aide à l'affectation des projets.

2

Organisation et choix techniques de réalisation de l'outil d'affectation

1 Résoudre le problème d'affectation

Il existe plusieurs manières de résoudre le problème d'affectation. Nous rappelons que le but est d'attribuer pour chaque étudiant un unique projet en maximisant la satisfaction générale.

Dans un premier temps, nous avons recherché des algorithmes permettant de résoudre ce problème. Nous verrons 2 méthodes, l'une utilisant un algorithme Hongrois et l'autre utilisant un solveur linéaire.

1.1 L'algorithme Hongrois, de Kuhn-Munkres

L'algorithme Hongrois permet de trouver un couplage parfait dans un graphe biparti de poids maximum (ou minimum) d'après Wikipédia.

Pourquoi est-ce un algorithme adapté au problème d'affectation ?

Un graphe biparti ? Pourquoi ?

Un graphe biparti un type particulier de graphe dont on distingue 2 nuages de noeuds dont les noeuds appartenant à un nuage ne peuvent être liés qu'aux noeuds de l'autre nuage par des arcs. Autrement dit, il est interdit d'avoir un arc entre 2 noeuds d'un même nuage.

On a une analogie directe entre les 2 nuages de noeuds du graphe, le premier nuage représente les étudiants et le second représente les projets.

Chaque étudiant est lié à chaque projet par n arcs pour les n projets. Par exemple dans la **Figure 1**, on dispose de 3 étudiants ayant émis une préférence pour chaque projet. La valeur de la préférence correspond au poids sur l'arc reliant l'étudiant au projet.

Un couplage parfait ? À quoi ça correspond ?

Un couplage parfait dans un graphe correspond à choisir des arcs dans ce graphe de façon à ce qu'il n'existe qu'un et un seul arc entre chaque noeuds. Dans notre cas, si on réalise un couplage parfait sur le graphe biparti, alors on aura réalisé une affectation des étudiants à un projet.

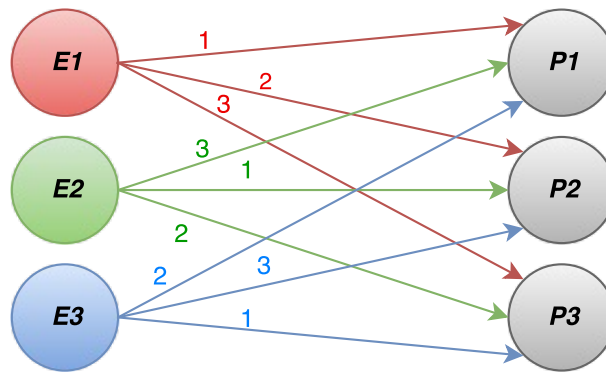


Figure 1 – Modélisation étudiants-projets dans un graphe biparti complet

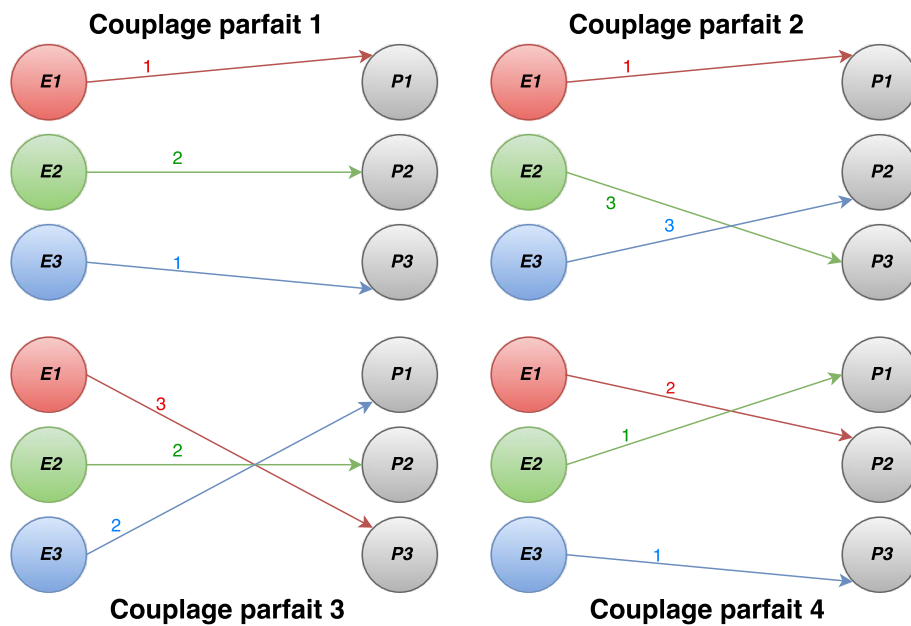


Figure 2 – Couplages parfait sur le graphe biparti (Figure 1)

La Figure 2 illustre ces propos en mettant en évidence quelques couplages possibles à partir du graphe biparti (Figure 1).

La méthode hongroise de résolution permet alors de trouver le couplage parfait maximisant (ou minimisant) la somme des poids de tous les arcs du couplage.

Pour nous, cela correspond à trouver les affectations qui satisfont tout le monde !

Nous ne rentrerons pas dans les détails de l'implémentation de l'algorithme hongrois car ce n'est pas la solution que nous avons retenu pour effectuer nos calculs d'affectation.

Nous l'avons toutefois testé lors du démarrage du projet en déroulant l'algorithme à la main sur un petit jeu de données et en utilisant une *librairie Munkres* implémentant la méthode hongroise sur *python* et *JavaScript*.

De la littérature très bien écrite est disponible sur Internet avec notamment : TODO AJOUTER LES REFERENCES

Nous nous sommes rapidement tourné vers l'optimisation par programmation linéaire plus flexible en termes de contraintes pour la résolution des problèmes et pour laquelle il existe de très puissants algorithmes implémentés dans des logiciels optimisés.

En effet, si résoudre le problème d'affectation comme le fait la méthode hongroise revient à résoudre le problème linéaire que nous avons défini au Chapitre 1.

La seule différence est que seules les contraintes **Equation 2** (Chapitre 1) et **Equation 1** (Chapitre 1) sur le nombre de projets par étudiants et le nombre d'étudiants par projet est fixé à 1 pour que la méthode hongroise soit applicable. De même la fonction objectif de ce problème sera forcément **Equation 3** (Chapitre 1). La complexité de cet algorithme est comprise entre $O(n^3)$ et $O(n^4)$, c'est à dire qu'elle s'exécute en un temps proportionnel au polynôme n^3 ou n^4 avec n la dimension de la matrice *affectation* à déterminer.

Le besoin de changer de méthode résolution et la nécessité de passer à la programmation linéaire est apparu lorsque nous avons commencé à ajouter des contraintes à notre modèle.

Nous allons voir désormais comment l'optimisation linéaire permet de résoudre le problème.

1.2 Résolution par un Solver Linéaire

Pour résoudre le problème d'affectation, après avoir testé la méthode Hongroise, nous avons découvert le solver GLPK qui permet de résoudre les problèmes d'optimisation linéaire.

Un **solver linéaire** est une machine à calculer équipée d'algorithmes d'optimisation permettant de trouver la solution à un modèle mathématique qui maximise (minimise, ou se rapproche d'une valeur donnée) au mieux la fonction objectif qui est combinaison linéaire des variables à déterminer.

Tel que nous l'avons défini, le problème d'affectation est **un problème d'optimisation linéaire**.

Ce qui peut rendre la résolution du problème compliqué est le fait que les **variables de décision**, dans notre cas les *affectations* doivent être des nombres entiers. En effet, un étudiant ne peut pas être affecté à 0,5 projet.

Comme il s'agit d'un problème dont les solutions sont des nombres entiers, nous avons utilisé la programmation linéaire en nombre entiers (PLNE) pour le résoudre.

Un problème de programmation linéaire en nombre entiers est construit de la même façon que dans le **Chapitre 1**.

Il faut fournir un jeu de **paramètres** au problème, **variables** de décisions, **contraintes** et enfin une **fonction objectif**.

Une façon naïve de résoudre le problème serait de procéder par itération en listant toutes les valeurs possibles des variables à déterminer et de les tester une par une.

Alors il suffira de voir si une valeur donnée respecte toutes les contraintes et obtient une meilleure note à la fonction objectif.

Si c'est le cas, alors on gardera cette solution jusqu'à en trouver une meilleure où lorsque l'on aura testé toutes les possibilités.

Il y a un gros problème de performances avec cette méthode, car pour notre problème, la matrice des *affectations* a une taille de $nbProjets \times nbIndividus$ et ne contient que des 0 et des 1.

Un calcul rapide nous permet de dire qu'il existe $2^{nbProjets \times nbIndividus}$ combinaisons de la matrice *affectations*. En prend $nbProjets = nbIndividus = n$, c'est à dire autant d'élèves que de projets.

La complexité de cette recherche de la meilleure solution est de $O(2^{n^2})$, ce qui est vraiment mauvais même pour de très petit échantillons. Le temps de résolution sera exponentiel en fonction de la taille des données d'entrée.

$$n = 4 \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \text{Exemple de matrice pour affectation} \quad (1)$$

Parmi les valeurs possible de la matrice d'affectations, nombreuses sont celles qui ne respecteront pas la contraintes d'un projet par étudiants ni même la contraintes d'un projet par étudiants.

C'est a dire qu'il n'existe qu'au mieux 1 seul 1 sur chaque ligne et chaque colonne de la matrice. On pourrait donc se servir de ces informations pour discriminer certaines des valeurs de la matrice *affectations*.

Pour une matrice de taille $n \times n$, on ne peut construire qu'un dictionnaire de $n!$ matrices potentiellement solution du problème d'affectation.

$$n = 4 \left\{ \begin{array}{l} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} & \leftarrow 4 = n \text{ positions dispo.} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} & \leftarrow 3 = n - 1 \text{ positions dispo.} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} & \leftarrow 2 = n - 2 \text{ positions dispo.} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} & \leftarrow 1 = n - 3 \text{ positions dispo.} \end{array} \right\} n! \text{ matrices possibles} \quad (2)$$

On peut désormais baser notre recherche itérative sur ces $n!$ matrices de la variable *affectation*. La complexité de la recherche est désormais de $O(n!)$. Par exemple pour une matrice 4×4 , une recherche itérative simple en testant toutes les combinaison amènerai à rechercher parmi $2^{4 \times 4} = 2^{16} = 65535$ matrices alors qu'après la discrimination des matrices ne pouvant pas être solutions, on a plus que $4! = 24$ matrices à tester.

Cette méthode est moins efficace que l'algorithme Hongrois qui s'exécute en $O(n^4)$ au pire.

Cependant cette première optimisation de la recherche de la meilleure affectation fait parti d'un des nombreux mécanismes d'optimisation utilisés par le solver.

Cet exemple était là pour illustrer la façon avec laquelle on peut rechercher la solution au problème en fonction d'un modèle donné.

Cependant, un solver est complexe et nous n'avons pas passé en revue toutes les méthodes d'optimisation pour notre problème.

Par la suite, nous vous présenterons le solver utilisé qui est GLPK, un solver linéaire avec des méthodes de résolution efficaces même pour de larges volumes de données.

2 Implémentation de la méthode de résolution

Pour mettre en application notre modèle mathématique, nous avons choisi d'utiliser le solver linéaire **GLPK** et d'écrire notre modèle en **GMPL** aussi connu sous le nom de **Mathprog**.

En effet, nous avons précédemment formalisé le modèle du problème d'affectation et il ne reste plus qu'à le traduire en langage informatique pour qu'un solver trouve une solution optimisée qui respecte toutes les contraintes.

Nous avons choisi GLPK car c'est un solver linéaire libre multi-plateforme qu'il est possible d'intégrer dans tout type d'applications à travers des librairies. L'avantage premier étant est qu'il est adapté à la résolution de notre problème linéaire.

Lors de ce projet, nous n'avons pas fait le tour de toutes les méthodes d'optimisations utilisées par les solveurs, nous avons surtout cherché à comprendre comment utiliser le solveur et le langage GMPL pour l'utiliser à la résolution de notre problème.

Le langage GMPL est un langage de programmation linéaire haut niveau permettant de mettre en forme un modèle mathématique très facilement pour qu'il soit résolu par le solveur GLPK.

Ce langage a été pensé pour être très proche du formalisme mathématique.

Il est possible avec GLPK de séparer données et modèle mathématique.

En effet, on peut disposer d'un fichier **[Mon_Modèle].mod** et lui associer des données **[Mes_Données].dat**

On peut ensuite appeler GLPK simplement via la commande :

```
glpsol -m [Mon_Modèle].mod -d [Mes_Données1].dat -d [Mes_Données2].dat
```

Vous trouverez en annexe l'implémentation du modèle mathématique. Dans ce modèle, nous avons implémenté l'ensemble du jeu de contraintes défini dans le début de ce rapport ainsi que déclaré l'ensemble de paramètres et variables nécessaires à la résolution (Cf. [Section 4](#)).

GLPK est disponible en librairie portable à cette adresse <https://www.gnu.org/software/glpk/>.

On peut exécuter notre modèle avec un jeu de données de tests intégrés via la commande :

```
glpsol -m Nom_du_Fichier.mod
```

Désormais, nous allons traiter le problème de la saisie des données et comment nous avons choisi de construire notre interface pour utiliser le modèle.

3 Interface de saisie des données et création d'un back-end

Dans cette partie du rapport, nous allons voir notre proposition en terme d'interface de saisie des préférences accompagnant notre modèle de résolution.

3.1 Les choix et les objectifs

Nous avons fait une étude sur les différentes possibilités de saisie (Excel, Google sheets, Celene, interface JAVA, console...). Après avoir pesé les pour et les contre, nous nous sommes tournés vers **Google sheets** : une application web (un tableur similaire au logiciel excel Microsoft) faisant partie de la suite de google. Une solution modulable permettant la saisie des vœux et des paramètres grâce à :

1. une mise en forme **flexible** et **évolutive** : l'outil tableur est très puissant ;
2. une saisie **efficace** et **collaborative** : plusieurs étudiants peuvent modifier le tableur en même temps ;
3. une mise en forme **intuitive** : affichage de notification et formulaire ;
4. une accessibilité **simple** : un produit accessible gratuitement, via un lien et modifiable directement sans avoir besoin d'être connecté. Aucune installation n'est requise ;
5. un langage de programmation **intuitif** : javascript ;
6. un **support actif** avec documentation **soignée** ;
7. une totale **liberté de communication** : appel d'api, lecture de fichiers, communication direct avec des pages html, création de modules.

À propos de la résolution, nous avons fait le choix d'utiliser la librairie **GLPK** disponible dans plusieurs langages de programmation dont javascript.

Avec des compétences en javascript et en connaissant le domaine des APIs, nous avons fait le choix de faire communiquer Google Sheets (l'interface de saisie) avec un serveur NodeJS

qui comporte le framework **Express** utilisé en temps qu'API (Interface de Programmation Applicative¹). Ceci nous a mené à une infrastructure d'application simple, moderne et puissante chargée de traiter les données envoyées par Google sheets. L'avantage principal de l'API est le fait qu'elle peut être réutilisée pour d'autres projets. Par exemple, si une application mobile est créée pour résoudre le problème d'affectation, celle-ci pourra communiquer avec l'API. En lui envoyant le bon format de donnée, celle-ci lui répondra avec les affectations.

En effet, Google Sheets a une interface de développement appelé "Google Script" permettant d'effectuer des algorithmes en javascript. C'est avec ce service que nous faisons appel à notre API.

La contrainte est que Google sheets ne communique pas avec des API local. Nous avons donc procédé à l'hébergement de cette application sur un domaine réservé : "testing-app.ovh/assign" et un serveur vps (serveur privé virtuel²).

Cette api génère les données du modèle mathématique et les lie au **langage GMPL** (MathProg) en utilise une librairie intégrée GLPK pour le résoudre. La réponse est renvoyée en JSON à Google Sheets qui lui, se charge de l'afficher.

On peut résumer le fonctionnement global de notre logiciel à travers **Figure 3**.

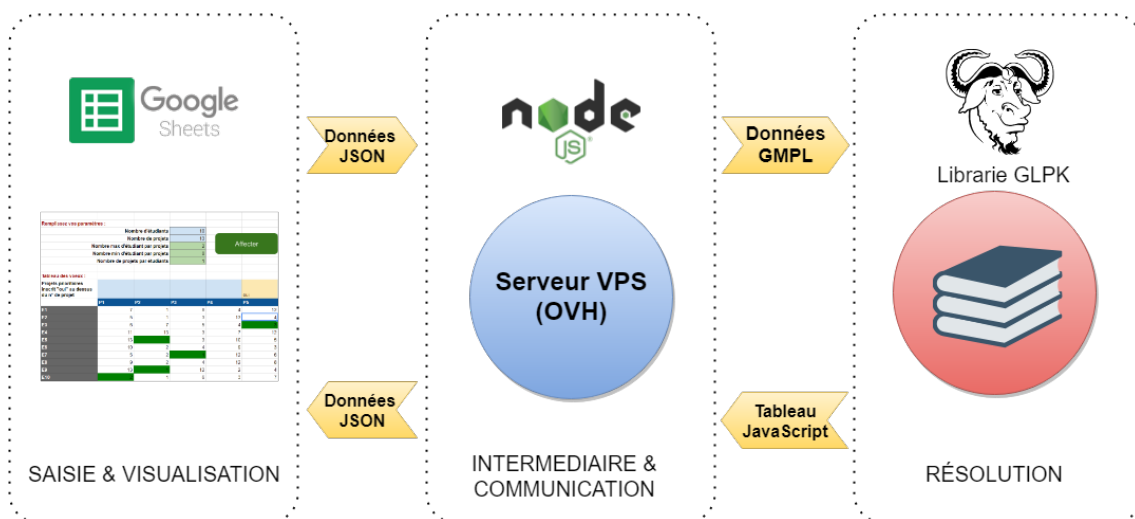


Figure 3 – Architecture de l'application de résolution des affectations

Sans plus attendre, nous allons entrer dans le vif du sujet et vous présenter l'interface que nous proposons.

3.2 Réalisation de l'interface de saisie

La **Figure 4** présente l'interface de l'application dans sa globalité. Elle dispose d'un tableau permettant à chaque élève/binôme ou groupe de pouvoir donner leurs vœux en fonction des projets (Cf. **Figure 5**). Les vœux vont de 1 (le plus fort) jusqu'au nombre de projets.

Nous avons cherché à produire une interface simple et minimaliste.

L'utilisateur pourra également configurer les paramètres d'affectation affichés en vert (Cf. **Figure 6**) :

1. Nombre max d'étudiant par projets

1. Une API est un logiciel utilisé pour résoudre un problème donné où répondre à un service permettant aux développeurs de ne pas se soucier de son implémentation.

2. Un VPS est une des nombreuses machine virtuelle hébergé sur une machine hôte puissante permettant d'avoir plusieurs serveurs sur une machine physique.

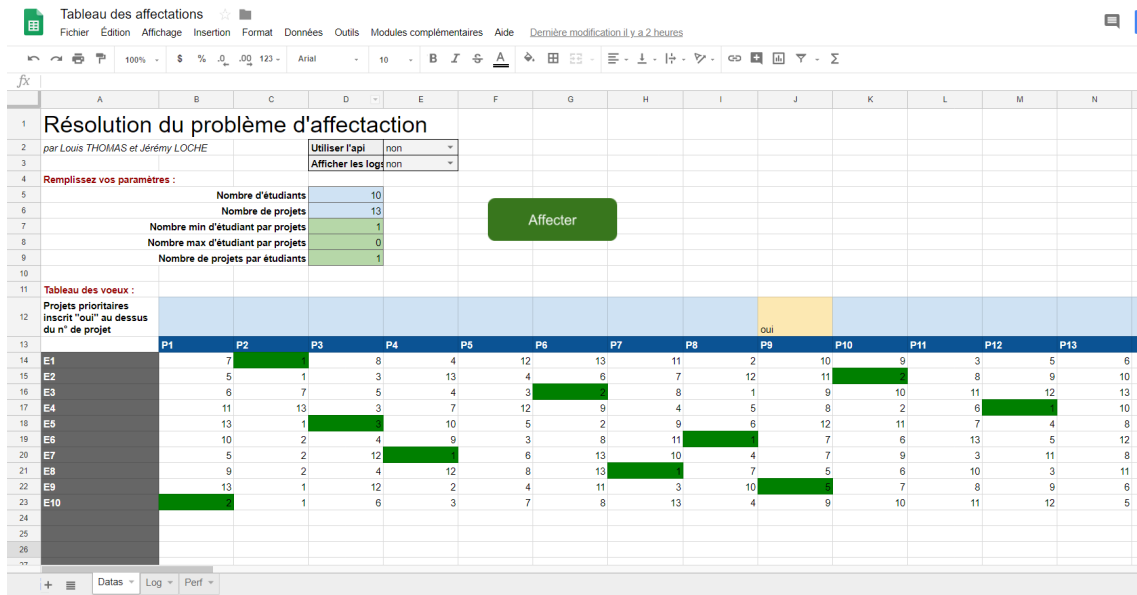


Figure 4 – Interface globale de l'application

	P1	P2	P3	P4	P5	P6	P7
E1	7	1	8	4	12	13	11
E2	5	1	3	13	4	6	7
E3	6	7	5	4	3	8	12
E4	11	13	3	7	12	9	4
E5	13	1	10	5	2	6	9
E6	10	2	4	9	3	8	11
E7	5	2	12	6	13	10	10
E8	9	2	4	12	8	13	1
E9	13	1	12	2	4	11	3
E10	2	1	6	3	7	8	13

Figure 5 – Tableau de saisie des préférences

2. Nombre de projets par étudiants

Nombre d'étudiants	10
Nombre de projets	13
Nombre max d'étudiant par projets	1
Nombre de projets par étudiants	1

Figure 6 – Saisies des paramètres d'affectation

Pour finir, l'interface comporte un bouton "Affecter" pouvant lancer le calcul de l'affectation des élèves aux projets proposés (Cf. Figure 7).



Figure 7 – Bouton affecter

À l'évènement du clic sur le bouton, une fonction interne à Google sheets est exécuté en arrière plan qui permet de récupérer les valeurs des paramètres et le contenu tableau pour envoyer une requêtes à l'API avec ces paramètres. Dès lors, l'API prend le relais et s'occupe de mettre en forme les données pour l'appel de GLPK et retourner les résultats d'affectation.

Dès la réponse intercepté, celle-ci est traité en affichant les affectations en vert sur le Google

Sheets (Cf. **Figure 5** pour une illustration du résultat).

Cette interface a permis à elle seule de résoudre le problème. Nous aurions pu nous arrêter à ce niveau car l'ensemble des objectifs du projet sont à ce stade accomplis.

Anecdote

Nous avons tout juste réussi à mettre en place cette version de l'outil d'affectation juste avant **les affectations de projets électroniques de DII3A**. Ceci a été l'occasion de mettre à l'épreuve notre modèle avec un réel jeu de donné. Ainsi tous les élèves de DII3 ont voté pour les projets électronique de leurs choix de manière collaborative en se connectant tous sur le Google Drive.

Nous avons ensuite lancé l'affectation via notre outil et le résultat a satisfait tout le monde ! Ce fut un franc succès !

Cependant, nous ne nous sommes pas arrêté à ce niveau mais nous avons mis en place un ensemble d'améliorations au programme.

3.3 Les améliorations

Après l'objectif principal réalisé. Nous nous sommes penché sur les améliorations nécessaires et utiles :

Amélioration de la fiabilité :

Google Sheets est actuellement dépendant de l'API pour fonctionner. De plus, l'API à besoin d'être hébergé (coûts) et peut s'arrêter à tout moment (ex : crash du serveur) ce qui rendrait l'application inutilisable. Pour remédier à cette dépendance, nous avons réussi à inclure la librairie javascript de GLPK dans Google Script et à l'utiliser. Par la suite, nous avons également réussi à importer le modèle mathématique. Ces étapes nous ont permis de rendre l'application indépendante de l'API. Ceci rends la duplication de notre application très facile et elle n'est dépendante que des services Google connus pour être redoutablement fiable.

Une interface plus intuitive

Les étudiants n'ont pas le droit d'appliquer le même vœux sur plusieurs projet différent. (ex : vœux n° 1 sur le projet 1 et vœux n° 1 sur le projet 2).

Afin que les problèmes soient visibles et rectifiés rapidement avant le début des affectations, nous avons mis en place une règle de mise en forme conditionnelle qui a pour but d'afficher en rouge les vœux qui ont la même préférence.

Par la suite, nous avons mis en place l'affichage d'une boîte de dialogue permettant de notifier l'utilisateur de la résolution en cours.

Il est affiché "*Initialisation du modèle ...*" : il s'agit de la récupération des données du tableau et la mise en forme de ces données afin qu'elles puissent être traité dans le modèle mathématique.

Par la suite, il y a "*Résolution en cours...*" : cela signifie que les données sont en train d'être traité par la librairie GLPK.

Pour finir, une boîte de dialogue affichera que la résolution a réussi avec le temps de la résolution, soit l'utilisateur sera informé d'une erreur (données incorrects, erreur 404/500...). Le temps de la résolution sera inscrit dans une feuille nommé "*Perf*" afin de pouvoir visualiser les différentes performance.

Une protection du code

Le code était accessible par tout le monde. Afin d'éviter des modifications consécutive ou voir une perte du fichier, nous avons mis à disposition le fichier original de base en mode "*Lecture seule*" et pouvant donc être copié pour être modifié.

Puis dans un second temps, nous avons créé un API Google Script avec le code principal. Cette API peut être accessible par le public. Autant dire qu'on peut totalement se passer d'héberger l'API sur un serveur VPS puisque le service Google le fait.

Une configuration plus poussée

On donne la possibilité à l'utilisateur de choisir de résoudre le problème avec ou sans l'API Node (l'API Node à l'avantage de résoudre le problème plus rapidement). Et il peut également choisir d'afficher les Logs dans une feuille nommé "*Log*".

L'ajout d'une option pour définir les projets prioritaires

Cette option permettra à l'utilisateur de définir les projets qui seront obligatoirement pris. Il suffira d'inscrire dans la case juste au dessus du projet : "*oui*". Nous avons rajouté cette contrainte car il est possible que plus de projets que d'étudiants soient proposés. Parmi cette liste de projets, certains peuvent être obligatoires, ce fut notamment le cas pour notre cas pratique (Cf. paragraphe [Section 3.2](#)). Cette option supplémentaire s'est traduit par une nouvelle contrainte dans le modèle mathématique. Pour ça on a besoin d'un nouveau paramètre qui est le *projetObligatoire* (Cf. [Equation 3](#)).

$$\forall p \in Projets, projetObligatoire[p] = \begin{cases} 1 & \text{si le projet est obligatoire} \\ 0 & \text{sinon} \end{cases} \quad (3)$$

$$\forall p \in Projets, projetObligatoire[p] \times \sum_{e \in Etudiants} affectation[e, p] \geq projetObligatoire[p] \quad (4)$$

L'[Equation 4](#) est la nouvelle contrainte résultante de l'ajout des projets obligatoires. Cette contrainte stipule qu'il faut **au moins** un étudiant affecté sur le projet s'il est obligatoire.

L'affichage des performances

Afin de connaître les performances de l'application, nous avons affichons sur une feuille "*Perf*", la vitesse de résolution en fonction de 2 paramètres : affichages de logs et utilisation de l'API Node ou non.



Conclusion

Ce projet nous a permis de découvrir les mécanismes de la programmation linéaire en nombre entier pour résoudre des problèmes d'optimisation.

Grâce à ce travail, nous avons pu mettre en place un modèle mathématique adapté à notre problème d'affectation des projets étudiants.

Ce modèle ensuite implémenté en GMPL et résolu par le solveur GLPK nous a permis de résoudre le problème avec un jeu de contraintes intéressantes.

Sur ce modèle nous avons greffé une interface de saisie intuitive basé sur Google Sheet.

Ce projet a été très satisfaisant pour nous car nous avons réussi à l'appliquer directement lors des affectations de projet électroniques de 3^e année en informatique industrielle.

De nombreuses améliorations ont été apporté à notre applicatif notamment sa mise en disponibilité en ligne pour une utilisation transparente. De plus, cette application est évolutive, aujourd'hui basée sur Google Sheet, elle pourra évoluer pour devenir une application PC ou mobile native.

Nous avons pu enrichir nos connaissances en recherche opérationnelle et en technologie web en produisant un outil fonctionnel permettant d'aider à la résolution des affectations de projets.

Annexes

4 Code GMPL pour le modèle mathématique d'affectation

```
1  /* Projet DII12 problème d'affectation aux projets étudiants , LOCHE Jérémy, THOMAS Louis */
2
3  /* Ce modèle mathématique est écrit en GNU Mathprog pour le solver linéaire GLPK
4
5     Il permet d'affecter des étudiants à des projets avec un petit jeu de paramètres.
6     L'utilisateur peut forcer où non l'affectation de projets dans le cas où plus de ↩
7        projets que d'étudiants sont dispo.
8
9     Ce modèle maximise la préférence des étudiants affectés à leur projets aussi maximise ↩
10        une variable servant de borne inférieure aux affectations.
11    Ainsi, on recherche la meilleure solution qui maximise à la fois la préférence mais ↩
12        offre une solution dont l'étendue est faible pour ajouter
13        cet effet de justice aux affectations.
14 */
15
16 param nbStudents, integer, > 0;
17 /* number of students */
18 /* Nombre d'étudiants */
19
20 param nbProjects, integer, > 0;
21 /* number of projects */
22 /* nombre de projets */
23
24 param maxStudentsPerProjects, integer, >0;
25 /* Maximum number of students per projects */
26 /* Nombre maxi d'étudiants par projets */
27
28 param ProjectsPerStudents, integer, >=0;
29 /* Nombre de projets par étudiants */
30
31 set I := 1..nbStudents;
32 /* set of students */
33 /* ensemble des étudiants */
34
35 set J := 1..nbProjects;
```

```

34 /* set of projects */
35 /* ensemble des projets */
36
37 param preference{i in I , j in J}, >= 0;
38 /* cost of allocating project j to student i */
39
40 param mandatoryProjects{j in J}, binary, >=0;
41 /* set of mandatory projects , 1 means mandatory project , 0 optional */
42 /* Ensemble des projets obligatoires , 1= obligatoirement affecté à au moins 1 étudiant , ←
    0=optionnel*/
43
44 var affectation{i in I, j in J}, binary, >= 0;
45 /* affectation[i,j] = 1 means projects j is assigned to student i
    note that variables affectation[i,j] are binary */
46 /* Variable de décision des affectation[i,j] = 1 correspond à projet j affecté à ←
    l'étudiant i, remarquez qu ces valeurs sont binaires*/
47
48
49 var minPreferenceValue, integer, >=0, <=nbProjects;
50 /* variable to optimize, represents the lowest preference value used for an assignement, ←
    this is to be maximised*/
51 /* Variable à optimiser , c'est la borne inférieure des préférences utilisés pour une ←
    affectation , elle sera maximisée*/
52
53 s.t. howManyProjectsPerStudents{i in I}: sum{j in J} affectation[i,j] = ProjectsPerStudents;
54 /* each student can only perform at most "ProjectsPerStudents" projects*/
55 /* Chaque étudiant ne peut faire au plus que "ProjectsPerStudents" projets*/
56
57 s.t. howManyStudentsPerProjectsMax{j in J}: sum{i in I} affectation[i,j] <= ←
    maxStudentsPerProjects;
58 /* each project must be assigned to at most maxStudentsPerProjects */
59
60 s.t. minPreference{i in I}: sum{j in J} affectation[i,j]*preference[i,j] >= ←
    minPreferenceValue;
61 /* each assigned student should be assigned with a preference higher than ←
    "minPreferenceValue"*/
62 /* chaque préférence utilisé pour chaque affectation doit être supérieure ou égale à ←
    "minPreferenceValue"*/
63
64 s.t. mandatoryProjectsST{j in J}: mandatoryProjects[j] * (sum{i in I} affectation[i,j]) ←
    >= mandatoryProjects[j];
65 /* each mandatory projects if it is one should be assigned to at least 1 student */
66 /* chaque projets obligatoire doit être affecté a au moins un étudiant */
67
68 maximize obj: minPreferenceValue + sum{i in I, j in J} preference[i,j] * ←
    affectation[i,j] ;
69
70 /* the objective is to find a best solution in term of mean preference and justice*/
71 /* La fonction objectif cherche la meilleur solution en terme de preference et de justice*/
72
73 solve;
74
75 printf "\n";
76 printf "Student\tProject-Number Preference\n";
77 printf{i in I} "%7d\t%7d\t%10g\n", i, sum{j in J} j * affectation[i,j],
78     sum{j in J} preference[i,j] * affectation[i,j];
79 printf "-----\n";
80 printf "          Total: %10g\n", sum{i in I, j in J} preference[i,j] * affectation[i,j];
81 printf "Min Preference: %10g\n",minPreferenceValue;
82 printf "Total Objective: %10g\n", obj;
83 printf "-----\n";
84 printf "Project\tNb Students\tMandatory\n";

```

```

85 printf {j in J} "%7d\t%7d\t%7d\n" , j, sum{i in I} affectation[i,j],mandatoryProjects[j];
86 printf "\n";
87
88 /* jeu de données exemple */
89 data;
90
91 param nbStudents :=10;
92
93 param nbProjects :=5;
94
95 param maxStudentsPerProjects := 2;
96
97 param ProjectsPerStudents := 1;
98
99 param mandatoryProjects :=
100 1 1
101 2 0
102 3 0
103 4 1
104 5 0
105 ;
106
107 param preference : 1 2 3 4 5 :=
108     1 1 5 3 2 4
109     2 1 2 3 5 4
110     3 5 2 3 1 4
111     4 3 4 2 1 5
112     5 2 3 1 4 5
113     6 3 5 4 2 1
114     7 4 2 3 5 1
115     8 2 3 1 4 5
116     9 3 4 5 1 2
117    10 5 1 2 3 4 ;
118
119 end;

```

Outil d'aide à l'affectation des projets

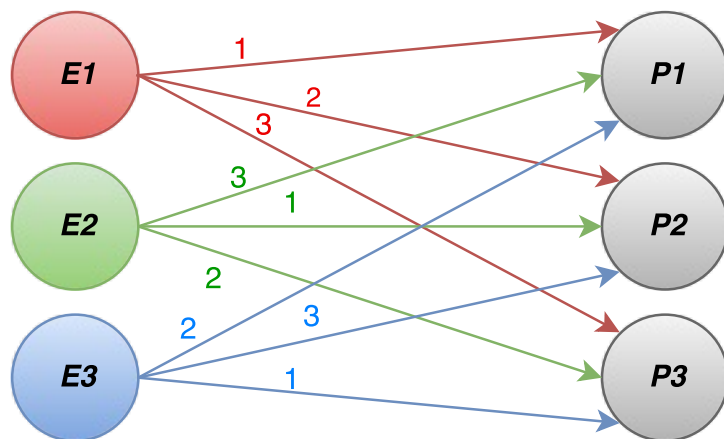
Jérémy Loche et Louis Thomas

Encadrement : Ameer Soukhal

Objectifs

Créer un outil permettant d'aider à la décision lors des intenses débats provoqués par les affectations de projets étudiants.

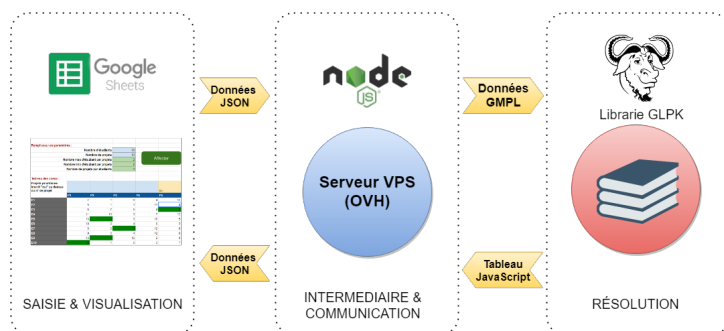
But : simplifier la phase d'affectation et satisfaire tout le monde avec le projet qui lui a été attribué !



3 étudiants pour 3 projets : Qui fait quoi ?

Mise en œuvre

Modélisation du problème d'affectation par un problème d'optimisation linéaire en nombre entier résolu à l'aide du **solver linéaire GLPK**. Saisie dans un tableur **Google Sheet** lié au solver GLPK via des scripts Javascript pour résoudre les affectations !



Architecture de l'outil

Résultats obtenus

Système complet avec un *modèle mathématique* implémenté en **GMPL** (pour GLPK) implanté dans une API en ligne pour faire appel au solver de manière abstraite. Et une interface de saisie simple.

Testé et approuvé par les étudiants de DII3 pour les affectations de projets électroniques !

Tableau des affectations

Résolution du problème d'affectation

par LOCH Jérémy et THOMAS Louis

Remplissez vos paramètres :

- Nombre d'étudiants : 3
- Nombre de projets : 3
- Nombre min d'étudiant par projet : 1
- Nombre min d'étudiant par projet : 1
- Nombre de projets par étudiants : 1

Tableau des votes :

Projet prioritaire

Export "tout" au-dessus du n° de projet

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
E1	1	2	3	4	5	6	7	8	9	10	11	12	13
E2	2	1	3	4	5	6	7	8	9	10	11	12	13
E3	3	2	1	4	5	6	7	8	9	10	11	12	13

Interface de saisie Google Sheet

Objectifs

Créer un outil permettant d'aider à la décision lors des intenses débats provoqués par les affectations de projets étudiants.

But : simplifier la phase d'affectation et satisfaire tout le monde avec le projet qui lui a été attribué !

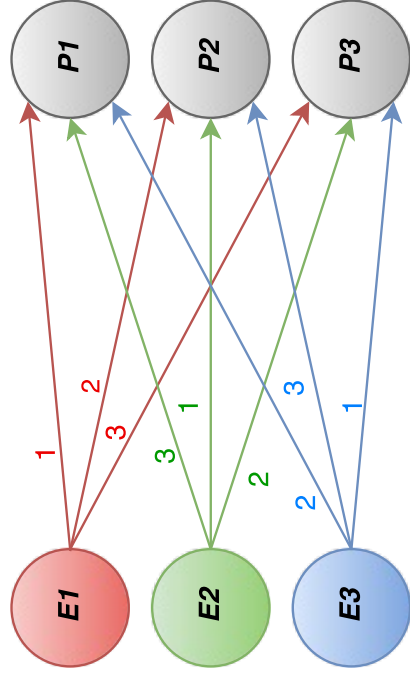
Mise en œuvre

Modélisation du problème d'affectation par un problème d'optimisation linéaire en nombre entier résolu à l'aide du **solver linéaire GLPK**. Saisie dans un **Tableau Google Sheet** lié au solver GLPK via des scripts Javascript pour résoudre les affectations !

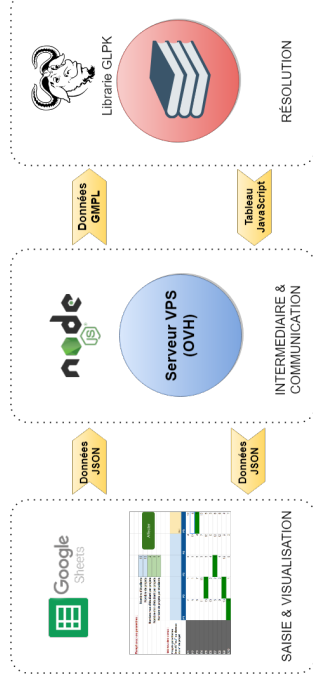
Résultats obtenus

Système complet avec un *modèle mathématique* implémenté en **GMPL** (pour **GLPK**) implémenté dans une API en ligne pour faire appel au solver de manière abstraite. Et une interface de saisie simple.

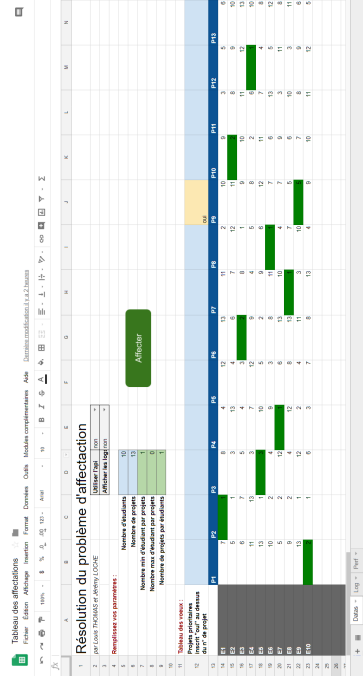
Testé et approuvé par les étudiants de DII3 pour les affectations de projets électroniques !



3 étudiants pour 3 projets : Qui fait quoi ?



Architecture de l'outil



Interface de saisie Google Sheet

Outil d'aide à l'affectation des projets

Résumé

Ce projet, vous présente une approche de la résolution du problème d'affectation des projets étudiants par l'optimisation linéaire. Vous découvrirez la programmation linéaire ainsi qu'une modélisation mathématique du problème et sa résolution à l'aide du solver linéaire GLPK. Vous comprendrez comment il est possible de modéliser le problème pour maximiser la satisfaction des étudiants et notre démarche dans la création d'un outil de saisie intuitif pour aider à résoudre vos problèmes d'affectations.

Mots-clés

affectation, solver, linéaire, recherche opérationnelle, optimisation, modèle, mathématique, Google, Sheets

Abstract

This project is all about finding the best suitable solution to the assignment problem which occurs when students have to choose a study project. You'll discover the mathematical modelisation of the problem and how it can be used to find the best solution to an assignment problem. You'll also discover our intuitive tool based on Google Sheets to solve your frustrating assignment problems.

Keywords

assignment, solver, linear, optimisation, Google, Sheets

Tuteur académique
Ameur SOUKHAL

Étudiants
Jérémy LOCHE (DII3)
Louis THOMAS (DII3)