Université de Technologie de Troyes - Automne 2020

Projet CL02

Sujet :

Optimization of product category allocation in multiple warehouses to minimize splitting of online supermarket customer orders

Mohammadmohsen AGHELINEJAD

Fait par :

Matthieu LEPICIER
Walid ALI HAIMOUD
Amel KEFI

Table des matières

1 Introduction

1.1 Mise en contexte

Dans le cadre de l'UE CL02 (Conditionnement, manutention et entreposage) nous sommes amenés à réaliser un projet par groupe de 4 à 5 étudiants. Ce projet consiste à étudier un article scientifique en détail, en comprendre les tenants et les aboutissants, notamment les méthodes utilisées et reproduire ces dernières dans l'optique d'obtenir des résultats proche de ceux des auteurs.

Dans un contexte de mondialisation et du développement du e-commerce, la gestion des coûts est devenue primordiale pour la compétitivité et la survie d'une entreprise. Cette gestion des coûts passe tout d'abord par une maîtrise totale de sa supply chain et sa gestion d'entrepôt. En effet, le choix de la localisation des entrepôts ainsi que de leurs contenus ne se fait pas au hasard. Plusieurs études sont faites en amont afin de s'assurer que ces choix soient bénéfiques en termes de gestion des coûts.

De plus à l'heure où l'emprunte écologique laissée par les supply chain représente un enjeu majeur, il est nécéssaire qu'une entreprise fasse son possible pour l'atténuer. Ainsi, l'opimisation du système de gestion de commande permet de réduire le nombre de camions sur les routes et donc les émissions de $C0_2$ qui en ressortent, mais également de réduire le plastique utilisé pour le packaging. Cela permet donc de séduire de nouveaux clients qui sont de plus en plus sensibles aux valeurs environnementales des entreprises.

Ce rapport présentera l'étude de l'article « **Optimization of product category allocation in multiple warehouses to minimize splitting of online supermarket customer orders** ». De la modélisation mathématiques du problème à l'analyse des résultats obtenus par les différentes méthodes et outils de résolution proposés.

1.2 Order Split Minimization Problem

Dans le e-commerce, nombreux sont les clients qui favorisent les sociétés offrant les frais de port. En effet, ces derniers peuvent représenter un coût non négligeable pour le client qui peut donc annuler sa commande au dernier moment. Les sondages le confirment : selon une étude menée par PayPal et ComScore, 43% des internautes français ne passent pas à l'acte d'achat, malgré la constitution d'un panier, à cause de frais de port trop élevés. Sondage confirmé par une étude de IFOP – Generix qui va même jusqu'à 55%. Nous pouvons donc dire qu'offrir les frais de port permet de se débarrasser du principal frein à l'achat en ligne. Afin d'être plus compétitif et d'attirer toujours plus de nouveaux clients, il est donc intéressant et judicieux pour une entreprise d'offrir ces coûts liés à la livraison. Mais attention, cela veut donc dire que ces derniers seront à la charge de l'entreprise, ce qui peut impacter la rentabilité et donc implicitement la pérennité de l'activité. La décision d'offrir les frais de livraison ne doit pas être prise à la légère et doit obligatoirement reposer sur une stratégie solide, aboutie et maîtrisée. Alors, comment offrir les frais de port tout en restant rentable ?

Dans le présent rapport, nous allons répondre à cette problématique en étudiant un des problèmes majeurs liés à la livraison dans le domaine du e-commerce. Ce problème est le « **splitting order** » (en Français : Division de commande). Ce phénomène intervient lorsqu'une commande regroupe plusieurs articles souvent de catégorie différentes stockés dans différents entrepôts. Dans ce cas, on ne peut éviter la division de commandes et donc la multiplication de livraisons entrepôts-client car plusieurs expéditions seront nécéssaires, augmentant par conséquent considérablement le coût de livraison et donc le coût global. Si ce phénomène n'est pas maîtrisé et étudié en amont par l'entreprise, deux cas de figures se dessinent :

• Il lui sera impossible d'offrir la livraison à ses clients et donc aura un manque à gagner non négligeable.
• L'entreprise sera moins rentable.

Les deux cas mènent inévitablement à une augmentation des coûts et une diminution de la satisfaction client, deux notions clés pour la survie d'une entreprise et son developpement.

L'exemple suivant permet de mieux visualiser le phénomène de splitting order auquel nous sommes confrontés :

• D'un côté, une entreprise propose sur son site internet des articles pouvant être regroupé dans 4 catégories différentes c_1, c_2, c_3, c_4 ($Fig.1$). Afin de stocker ces produits, l'entreprise dispose de 3 espaces de stockages : w_1, w_2, w_3 (une catégorie ne peut être stocké que dans un seul entrepôt).

• De l'autre côté, un client qui commande 4 articles de 4 catégories différentes c_1, c_2, c_3, c_4.

Ces articles n'étant pas stockés dans le même entrepôt le splitting order est inévitable : la commande initial est divisée en 3 sous-commandes (S_1, S_2, S_3) entraînant 3 livraisons à la charge de l'entreprise, dans le cas où l'entreprise offre les frais de port. Par contre, si l'entreprise avait décidé de stocker la catégorie d'article c_1 dans l'entrepôt 3, la commande initiale n'aurait été divisée qu'en 2 sous commandes (S_2, S_3). L'entreprise aurait donc économisé une livraison.

Au travers de cet exemple nous avons pu constater l'intérêt et l'importance du choix d'affectation des catégories de produits aux entrepôts. L'objectif est d'optimiser l'affectation de chaque catégorie de produit aux entrepôts, afin de minimiser le splitting order et donc implicitement minimiser les coûts de livraison et packaging.

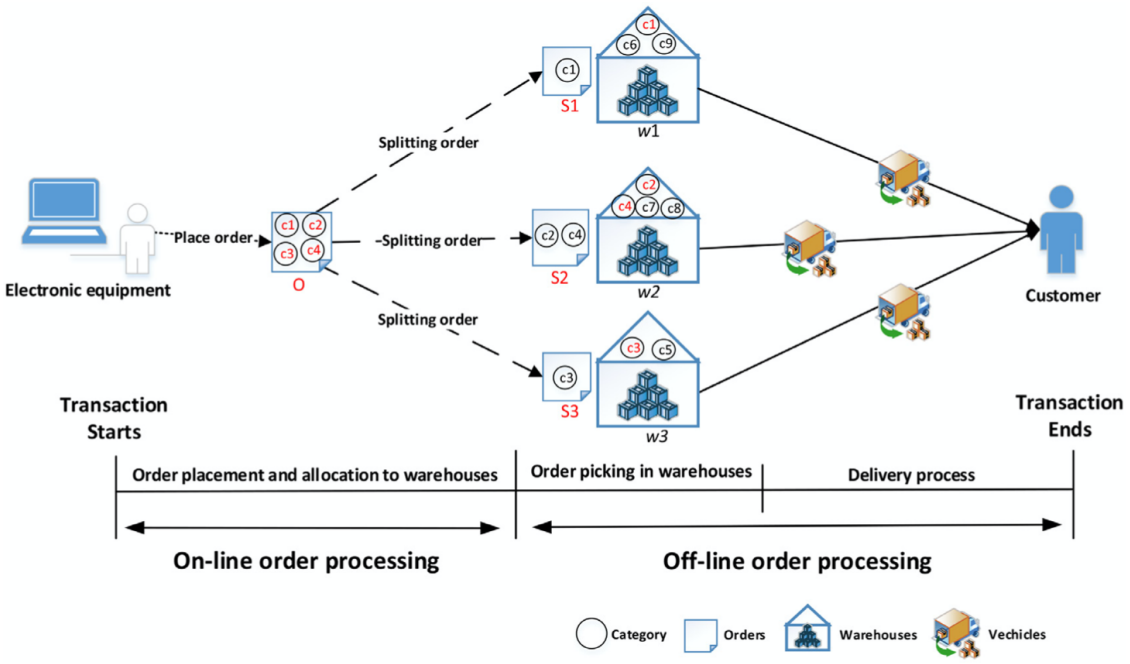

FIGURE 1 – Exemple de Splitting Order

1.3 Revue de littérature

Le problème de splitting order interesse grandement certains chercheurs qui ont proposé des modelisations mathématiques et des méthodes de résolutions exactes ou non pour ce sujet. Ainsi *Xu et al. (2005)* ont réalisé que la performance des systèmes d'execution de commandes des retailers était assez moyenne. Ils ont été les premiers à proposer un modèle d'execution de commande pour les e-commerce et ont souligné l'importance de minimiser les fractionnements des commandes "multi-produits" dans le but de réduire les coûts d'expedition. Ils ont construit une heuristique de recherche de voisinage pour évaluer l'affectation des commandes en temps réel. En 2009, *Xu et al.* ont cette fois intégré la prévision de la demande à leur précédent modèle. Ils ont alors construit un modèle de contrôle stochastique et ont proposé deux heuristiques. L'une d'entre elle a pour objectif de minimiser le coût total des livraisons des commandes clients notamment en minimisant les split des commandes multi-produits. En 2009 toujours, *Mahar et Wright* ont adopté une autre approche pour la résolution de ce problème. Au lieu de simuler un système d'execution de commande en temps réel, ils ont proposé que les commandes soient accumulées avant d'être assignée à un entrepôt. Ils ont ainsi developpé selon leur dires une politique d'affectation de commandes aux entrepôts "quasi-dynamique" qui répartie les commandes aux entrepôts en fonction des stocks prévus, des expeditions et du temps d'attente client pour accroitre la rentabilité de l'entreprise. Du reste en 2015, *Torabi, Hassini et Jeihoonian*, compte tenu de la dimension temporelle très présente dans ce problème pour la prise de décision, ont proposé une approche basée sur la décomposition de *Bender* pour trouver un système d'execution de commande affectant de façon optimale les commandes clients à ce qu'ils appellent des "centres d'execution" (*ie* entrepôts) et permettant si besoin d'établir des transferts de stocks entre les différents entrepôts. Enfin en 2012 puis en 2014 *Acimovic et Acimovic & Graves* ont abordé ce problème de splitting order en considérant le coût de renoncement des commandes futures epuisant potentiellement les stocks des entrepôts. Ils ont ainsi utilisé les valeurs duales d'un programme linéaire de transport pour créer une méthode asymptomatiquement optimale d'estimation des coûts d'expedition prévus.

Tous les articles cités precédemment ont le même point commun : Ils supposent tous que le type de produit dans chaque entrepôt est donné et que les split des commandes sont le résultat de ruptures de stock sur certaines références ne pouvant être reapprovisionnées à temps. Or on sait maintenant que dans le cas du e-commerce, bien que ces ruptures de stock existent, elles sont plutôt rares. En effet, en général pour un e-commerce disposant d'entrepôts, les responsables de ces derniers veillent toujours à maintenir les stocks à des niveaux sûrs et peuvent se le permettre car les coûts de possessions sont relativement faibles. Partant de ce postulat, d'autres chercheurs se sont rendu compte que pour le cas des e-commerce, on avait souvent dans une même commande plusieurs types de produits stockés dans des entrepôts différents et ont emit l'hypothèse que les split de commandes venaient de là et non des ruptures de stocks.

Ainsi *Catalan et Fisher (2012)* ont analysé le problème d'affectation des SKU aux différents entrepôts pour minimiser les divisions de commandes. Ils ont proposé quatre méthodes heuristiques et ont démontré à travers un exemple concret de e-commerce que leur heuristique *Bestsellers* était la plus performante pour ce problème. En effet, cette dernière s'est avérée être la plus efficace pour minimiser le split des commandes et a donné de très bons résultats au niveau de la flexibilité des affectation et de l'équilibre des stocks pour chaque entrepôt.

Finalement, avec la démocratisation de l'intelligence artificielle et notamment du machine learning ces dernieres années, de nouveaux outils et de nouvelles techniques sont apparues. Les algorithmes de clustering en font parti et ont beaucoup inspiré certains chercheurs dans le domaine de l'optimisation. En effet, un algorithme de clustering, qu'il soit supervisé (nécéssite des labels sur les données) ou non supervisé va permettre de retrouver pour des données des corrélations, des liens et de classifier les données en cluster. Si on se base sur cette définition, l'algorithme s'avère très utile dans des domaines transverses au machine learning comme l'optimisation et la recherche opérationnelle. Dans le cas de notre problème de split de commandes des chercheurs se sont inspirés du clustering pour construire une heuristique. Il s'agit de l'article que nous avons choisit d'etudier. Ainsi, *Zhu, Hu, Huang et al (2020)* ont proposé une résolution du problème de splitting order grâce à une heuristique basée sur le clustering qu'ils ont nommée *K-Links clustering algorithm (KLCA)*, leur objectif etant de minimiser le split de commandes et donc le nombre d'expeditions grâce à une affectation appropriées et faite en amont des catégories de produit dans les entrepôts. Pour cela ils ont décidé simplement de définir les similitudes entre deux catégories de produit comme des liens qui sont representés par le nombre de commandes qui contiennent les produits de deux catégories. Ils regroupent ainsi les catégories similaires dans les mêmes entrepôts en fonction des liens afin de minimiser le splitting order. Dans le présent rapport, nous allons étudier étudier plus en détail le modèle de PLNE en 0 - 1 qu'ils ont proposé.

1.4 Data Structure

Pour traiter ce problème, nous avons besoin d'un jeu de données comprenant un ensemble de commandes clients I multi-catégories. Les auteurs de l'article utilisent pour leur résolution 3 différents sets de données. Le premier provient de données réelles d'un e-commerce. Il contient 31 551 commandes multi-catégories de 96 différentes catégories et a uniquement servi à vérifier l'efficacité de l'heuristique. Ils ont ensuite généré aléatoirement 2 sets de données à plus petite échelle basée sur les probabilités de distributions observées dans les données réelles du e-commerce. Ils comprennent chacun 900 commandes clients et 12 catégories de produits.
Nous n'avons cependant pas été en mesure de traiter notre problème avec les données de l'article. Nous avons donc importé un dataset depuis la plateforme Kaggle. Il comprend 22 061 commandes clients et 60 catégories. La taille de notre dataset est en revanche trop importante pour être traité sur CPLEX. Il a alors fallu modifier ce dataset et l'alléger. Pour la résolution de notre problème, nous avons utilisé un dataset comprenant 117 commandes clients pouvant contenir jusqu'à 15 catégories de produits différentes.

Nous allons maintenant analyser la structure de nos données. On observe que 86.32% de nos commandes clients sont des commandes multi-catégories. En effet, comme on peut le voir sur la Figure 2, seulement 13.68% des commandes sont des commandes single-catégories. Parmi nos commandes multi-categories, on observe que le nombre de catégories par commande les plus réccurents dans une commande o_i sont : 2 catégories par commande avec 11.11% du dataset, 10 catégories avec 11.11% et 15 catégories avec 12%. Ainsi, le nombre de commandes single-catégorie est négligeable par rapport à celui des commandes multi-catégories.

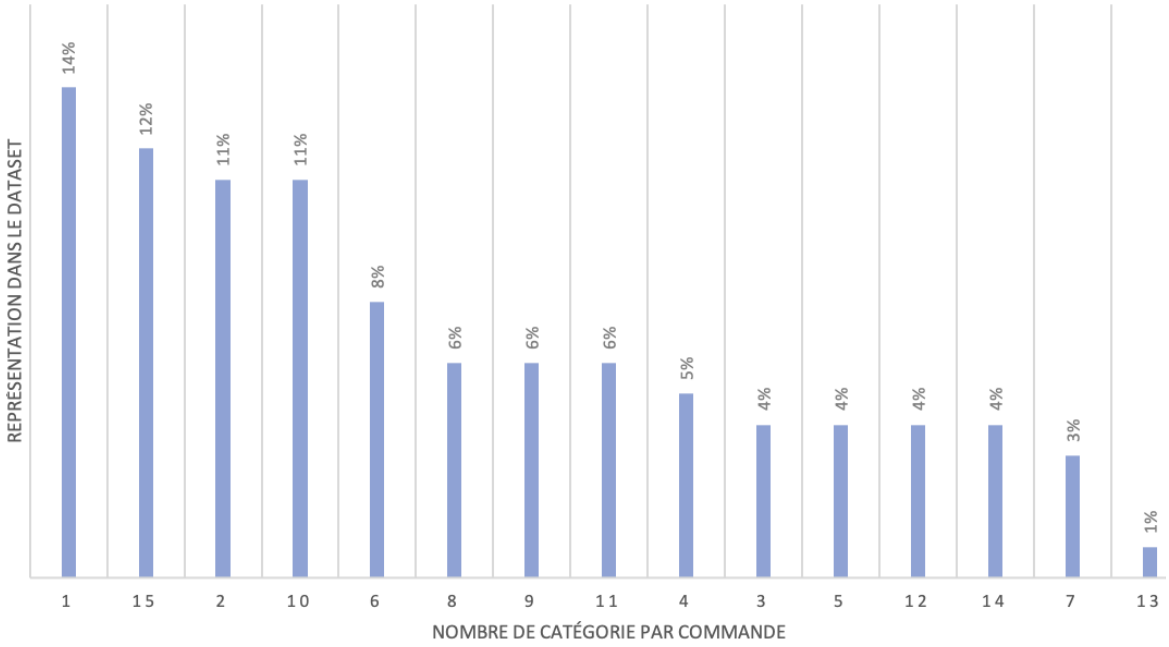

FIGURE 2 – Répartition de la typologie de commande en fonction du nombre de catégorie

Pour finir, nous allons analyser la matrice de liens entre les catégories $L(c_j, c_q)$ obtenue par rapport au nombre de commandes dans lesquelles deux catégories c_j et c_q apparaissent simultanément.

C	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_1	0.00	6.43	7.74	7.83	6.79	7.17	7.67	6.81	7.48	7.20	6.24	8.80	7.99	8.72	9.35
c_2		0.00	7.42	6.49	6.59	6.07	6.19	5.06	7.35	6.29	6.67	8.47	8.19	7.48	8.99
c_3			0.00	6.39	8.36	6.40	7.50	6.90	7.67	7.18	6.18	8.57	6.67	8.64	7.88
c_4				0.00	6.78	6.52	6.53	5.73	6.39	6.77	5.20	7.82	6.72	7.36	7.16
c_5					0.00	6.83	7.40	4.64	6.81	5.06	5.99	8.67	7.06	7.63	8.40
c_6						0.00	7.06	7.70	6.62	7.33	5.47	8.36	5.80	7.91	8.44
c_7							0.00	8.04	6.51	8.86	5.58	9.22	7.34	7.97	7.57
c_8								0.00	5.97	7.10	4.48	7.43	5.53	5.53	6.50
c_9									0.00	5.36	5.24	7.07	8.09	7.68	9.23
c_{10}										0.00	5.82	7.36	6.84	6.90	7.29
c_{11}											0.00	7.24	6.24	6.13	7.60
c_{12}												0.00	8.17	8.19	8.36
c_{13}													0.00	7.07	8.81
c_{14}														0.00	9.72
c_{15}															0.00

On observe que toutes les catégories sont liées. En effet, pour toutes catégories c_j et c_q, on a $L(c_j, c_q) > 0$. Cela veut donc dire que l'ensemble des catégories sont liées et qu'il est donc pertinent d'utiliser ce dataset. En effet pour une catégorie j donnée il existe au minimum un order i tel que pour toutes les autres catégories q différentes de j, j et q soient dans cet order. Les principes mathématiques qui ont permit de construire cette matrice $L()$ seront expliqués plus tard dans le présent rapport. Lors du calcul de la matrice de liens, les commandes single-catégorie ne sont pas prises en compte. Ils n'ont donc pas d'impacts sur le choix de distributions des catégories puisqu'ils n'engendrent aucun split.

2 Modélisation

2.1 Objectif et Hypothèse

Tout d'abord, nous parlons ici de problème lié à l'affectation de catégories de produits dans des entrepôts plutôt que de problème lié à l'affectation de SKU dans des lieux de stockages. Ce choix est cohérent puisque les entreprises ont pour politique de stocker les produits de même catégorie au même endroit. Cela facilite le picking, et permet également de réduire les coûts lors du remplissage des entrepôts. Cela veut donc dire que deux produits appartenant à une même catégorie ne peuvent pas être stockés dans deux entrepôts différents.

Dans l'approche de ce problème nous allons utiliser un algorithme de clustering pour l'assignement des catégories de produits dans les differents dépôts. Pour cela on définiera la similarité entre deux produits comme un lien, le lien est obtenu par rapport au nombre de commande dans lesquelles les deux produits apparaissent. Le principe est de regrouper les clusters sous le même entrepot. La répartition optimale des catégories entre les entrepôts doit être déterminée en fonction de la répartition globale des catégories de toutes les commandes multi-catégories. Il est donc nécéssaire d'avoir un nombre de données conséquent pour trouver une solution optimale, fonctionnelle et représentative de la réalité.

Afin de modéliser le problème, nous avons mit en place plusieurs hypothèses. Tout d'abord, le nombre d'entrepôts K doit être supérieur ou égale à 2, car s'il n y a un seul entrepôt, le problème n'existe pas : tous les articles seront forcément expédiés d'un seul et même endroit. Cette hypothèse revient à dire qu'il existe une capacité de stockage limité pour chaque entrepôt k ; un seul entrepôt ne peut stocker toutes les catégories de produits.

De même, si une commande client ne contient qu'une seule catégorie de produit, la commande sera forcément expédié d'un seul entrepôt, il n'y aura donc pas de splitting order. Afin de mieux visualiser la problématique, nous considérons uniquement les commandes contenant plusieurs catégories. On suppose également que le transporteur qui réalise la livraison entrepôt-clients n'a acune contrainte de capacité,c'est à dire qu'il est capable de transporter tout type de colis peu importe le volume et le poids. Cette hypothèse est importante car si le transporteur est limité, l'affectation des catégories de produits aux entrepôts ne sera peut-être pas la même : le splitting order global ne sera peut-être pas optimal. En effet, si le transporteur est limité, il est possible qu'il y ai des splitting order au niveau de la livraison depuis un même entrepôt.

Pour résoudre notre modèle nous avons utilisé deux dataset différents trouvé sur le site Kaggle. Un conséquent avec plus de 10 000 commandes, et un plus raisonable de 117 commandes pouvant tourner facilement sur nos machines.

Enfin, notre objectif étant de minimiser le nombre de splitting order dans le cas d'un e-commerce, nous excluons toutes les problématiques liées aux ruptures de stocks : nous supposons que les entrepôts ont suffisamment de stock pour satisfaire toutes les commandes.

2.2 Variables de décision et paramètres

Pour modéliser mathématiquement la situation, il est nécessaire d'adopter certaines notations pour les données, les paramètres et les variables de décisions.

Pour simuler ce problème nous utiliserons ainsi les formulations suivantes : .

i : indice représentant la commande d'un client avec $i = 1, 2, ..., I$
j : indice représentant les differentes catégories de produits existantes avec $j = 1, 2, ..., J$
k : indice représentant les differents entrepôts existants avec $k = 1, 2, ..., K$
o_{ij} : affectation commande - catégorie si la commande i contient la catégorie j alors $o_{ij} = 1$ sinon 0.

Puis les paramètres :

N_{max} : Nombre maximum de catégories pouvant être stocké dans un entrepôt avec $N_{max}<J$
N_{min} : Nombre minimum de catégories devant être stocké dans un entrepôt .
I : Nombre de commandes client .
J : Nombre de catégories de produits .
K : Nombre d'entrepôts .

L'objectif du modèle est de nous donner les affectations optimales des catégories aux entrepôts qui permettront de minimiser les splittings orders. Les variables de décisions sont donc les suivantes :

x_{jk} : binaire égal à 1 si la catégorie j est stockée dans l'entrepôt k , 0 sinon.
y_{ik} : binaire égal à 1 si la commande i est expédiée à partir de l'entrepôt k , 0 sinon.

2.3 Formulation du modèle mathématique

Maintenant que nous avons déclaré toutes les notations, nous pouvons modéliser le problème.

$$\text{Minimize} \sum_{i=1}^{I}(\sum_{k=1}^{K} y_{ik} - 1) \tag{1}$$

subject to :

$$\sum_{k=1}^{K} x_{jk} = 1, \forall j \tag{2}$$

$$\sum_{j=1}^{J} x_{jk} \leq \mathrm{N}max, \forall k \tag{3}$$

$$\sum_{j=1}^{J} x_{jk} \geq Nmin, \forall k \tag{4}$$

$$y_{ik} * J \geq \sum_{j=1}^{J} x_{jk} * o_{ij}, \forall i, k \tag{5}$$

$$x_{jk} \in \{1, 0\}, \forall j, k \tag{6}$$

$$y_{ik} \in \{1, 0\}, \forall i, k \tag{7}$$

1. `Fonction objectif` : La somme des y_{ik} sur tous les entrepôts k va correspondre au nombre d'entrepôts qui vont être utilisés pour satisfaire la commande i. Partant du fait que dans tous les cas, il y aura toujours au minimum un entrepôt utilisé, le nombre de 'split' correspondra au nombre d'entrepôts utilisés - 1. On répète cela pour toutes les commandes i, ce qui revient donc à minimiser pour toutes les commandes l'ensemble des splits
2. `Contrainte 1` : On impose le fait que chaque catégorie j soit entreposée dans un seul et unique entrepôt k. Cela nous évite de nombreux problèmes notamment le fait de devoir choisir entre différents entrepôt pour une même commande (cf Objectif et Hypothèses).
3. `Contrainte 2` : Contrôle la capacité de chaque entrepôt, chaque entrepôt doit avoir au maximum $Nmax$ catégories de produits.
4. `Contrainte 3` : Contrôle la capacité de chaque entrepôt, chaque entrepôt doit avoir au minimum $Nmin$ catégories de produits.
5. `Contrainte 4` : Pour chaque commande i, on regarde si l'entrepôt k est utilisé pour satisfaire la commande. Pour cela, on force y_{ik} à 1 (grâce à la multiplication par J) si la commande i contient au moins un produit qui appartient à une des catégories j stockées dans l'entrepôt k. La multiplication par J correspond au "Big M" que l'on retrouve souvent dans des modèles mathématiques. Le nombre maximal de catégories stockées dans un entrepôt étant $Nmax$ tel que $Nmax < J$, J est suffisant pour contraindre le système.
6. `Contrainte 5` : Contrainte de binarité sur la variable binaire d'affectation catégorie-entrepôt. Si une catégorie de produit j est entreposée dans un entrepôt k alors x_{jk}=1, 0 sinon.
7. `Contrainte 6` : Contrainte de binarité sur la variable binaire d'affectation commande-entrepôt. Si une commande i est expédiée à partir d'un entrepôt k alors y_{ik}=1, 0 sinon.

Ce modéle une fois lançé sur Cplex, nous permet de ressortir le nombre de Splitting orders optimal par rapport à nos données. Mais également l'affectation de chaque catégorie de produit dans les entrepôts.

3 Méthode de résolution

3.1 Programmation linéaire

Une fois le modèle mathématique defini, nous sommes passés à la résolution de celui-ci, ayant pour but de minimiser le nombre de Splitting orders. Pour cela, il nous a fallu utiliser un solveur capable de résoudre un modèle mathématique linéaire prenant en compte des contraintes. Nous avons donc décidé de travailler sur CPLEX. Il s'agit d'un solveur capable de résoudre des problèmes d'optimisation complexes que l'ensemble du groupe maîtrise et utilisé par les auteurs de l'article.

Un projet CPLEX est composé d'un fichier contenant le modèle mathématique, ainsi qu'un fichier de données. La première étape de l'élaboration d'un tel projet est donc la déclaration du modèle mathématique à travers les paramètres du problème ainsi que les variables de décisions, la fonction objectif et les différentes contraintes. Afin de faciliter l'utilisation des variables, nous avons défini une expression $Split = \sum_{i=1}^{I}(\sum_{k=1}^{K} y_{ik} - 1)$ permettant de définir le nombre de Splitting orders total que l'on utilisera dans la fonction objectif.

Pour procéder à la résolution, il est impératif d'alimenter le projet en données. Nous avons utilisé celles présentées dans la section 1.4 du rapport. En revanche, il s'agit là de données brutes qui ne correspondent pas aux données du modèle linéaire. En effet, pour traiter ce problème, nous avons besoin de la matrice o_{ij}. Nos données de base incluent seulement une liste des numéros de commandes et leurs categories. Il est alors nécessaire de procéder à une étape dite de *pre-processing* afin de traiter la donnée brute une première fois pour l'adapter à notre modèle mathématique. Il est possible de réaliser cette étape sur CPLEX mais nous avons choisi de retravailler ce jeu de données à l'aide d'une macro VBA permettant de construire la matrice o_{ij} à partir du dataset et notamment d'afficher cette derniere sur Excel de sorte à ce que le language OPL permette la lecture facile de la donnée.

La dernière étape de la résolution consiste en l'affichage des résultats afin de pouvoir les analyser. Les données à afficher sont alors les différents paramètres utilisés pour résoudre le problème à savoir K, le nombre d'entrepôts ouverts et leurs capacités N_{min} et N_{max}. Nous affichons également la variable de décisions x_{jk} et l'expression $Split$ qui vont nous permettre de savoir dans quel entrepôt les catégories sont assignées, ainsi que le nombre total de Splitting orders nécessaire pour assurer toutes les commandes. Pour finir, il nous a paru pertinent d'afficher le temps de compilation afin de comparer par la suite la performance des différents modèles sur les différents outils utilisés.

3.2 Transformation du problème

Dans ce premier modèle, l'objectif est de trouver une répartition optimale des catégories de produits dans les entrepôts permettant de réduire le nombre de Splitting orders. Le solveur va tester différentes combinaisons dans le but de minimiser le nombre de colis à envoyer. Il s'agit là d'un problème NP-complet. Ainsi, dès lors que la taille des données augmente ou que certains paramètres changent, tels que le nombre K d'entrepôts ouverts ou les capacités $Nmin$ et $Nmax$, on observe un temps d'exécution très important. Il devient donc pertinent de mettre en place une heuristique qui va nous permettre de contrer ce problème en obtenant une solution proche de l'optimum en un temps

d'exécution raisonnable. Cette heuristique ne se basera cependant pas sur le premier modèle.

Il est possible de voir le problème sous un angle différent en incluant la notion de clusters. Un cluster est un groupement d'éléments comportant des similitudes ou liens. On peut alors baser la répartition des catégories dans les différents entrepôts sur l'étude des liens entre les catégories présentes dans un set de commandes. On parle désormais de K-links Clustering Problem. On va définir le concept de lien $l_i(c_k, c_j)$ entre les catégories c_k et c_j comme étant égal à 1 si celles-ci sont incluses dans une order o_i, sinon égal à 0. Le lien $l_i(c_k, c_j)$ peut être vu comme un *inlink* ou un *outlink*. Autrement dit, si ces catégories sont dans un même dépôt, $l_i(c_k, c_j)$ sera perçu comme un *inlink* qui ne causera pas un order splitting. Au contraire, si ces categories ne sont pas dans le même dépôt, $l_i(c_k, c_j)$ sera perçu comme un *outlink* qui causera un order splitting. On observe donc une relation entre ce lien $l_i(c_k, c_j)$ et le problème de Splitting orders.

À partir de cela, on va procéder à une transformation du problème mathématique et de sa fonction objectif. On introduit une variable LW qui traduit la somme des *outlinks* (ou *inlinks*) entre les différents dépôts qui va influer sur le nombre de Splitting orders. Ainsi, on va chercher à réduire les Splitting orders en minimisant LW dans le cas où l'on considère les *outlinks*, ou maximiser LW si l'on considère les *inlinks*.

3.2.1 Formulation du modèle mathématique de l'heuristique

Étant donné qu'une transformation du modèle mathématique a été faite, il est important de définir les nouveaux paramètres et variables de décisions du problème. Comme expliqué précédemment, nous prenons désormais en compte les liens entre les catégories $l_i(c_k, c_j)$ pour définir la répartition des catégories dans les dépôts qui nous permettra d'évaluer les *outlinks* et *inlinks*. Dans cette partie, nous considèrerons uniquement les *outlinks*. L'étude des *inlinks* est équivalente. Pour comprendre comment le modèle fonctionne, nous allons passer en revue les différents paramètres et variables du problème.

Variable de décision :

Pour commencer, nous allons définir notre variable de décision. Comme dans le premier modèle, nous cherchons à affecter les catégories dans différents dépôts. Nous avons donc besoin de définir une variable booléenne x_{jk} qui prend 1 lorsqu'une catégorie c_j est affectée à un dépôt w_k, sinon vaut 0.

Paramètres du problème :

La somme pondérée des liens entre les catégories c_j et c_q pour toutes les commandes clients $L(c_j, c_q)$ est une des données importantes du problème. La matrice $L(c_j, c_q)$ représente l'affectation des catégories et leurs associations pour toutes les commandes clients. On la définit comme suit :

$$L(c_j, c_q) = \sum_{i=1}^{I} l_i(c_j, c_q) = \sum_{i=1}^{I} o_i(c_j) o_i(c_q) \theta_i, \forall i \in I, j \neq q, q, j \in J \tag{8}$$

On obtient cette équation en sommant sur i la multiplication de la matrice d'affection d'une commande o_i des catégories c_j et c_q par un coefficient de connection θ_i appelé contribution effect. La somme des

$l_i(c_k, c_j)$ sur toutes les commandes o_i sans inclure θ_i nous donne une surestimation trop importante des ordrers splitting. En effet, si on prend une commande avec deux catégories, le nombre de liens vaut 1, ce qui peut causer un seul split. Dès lors que nous avons plus de deux catégories dans une commande, le nombre de liens peut être supérieur au nombre maximum de splits possible. Il est donc nécessaire d'ajouter un coefficient de connexion θ_i pour satisfaire la condition suivante : pour chaque commande, la somme pondérée des liens est égale au nombre maximum de splits. On définit donc ce coefficient comme suit :

$$\theta_i = (NCO_i - 1)/C(NCO_i, 2), \forall i \;\; 0 < \theta_i \leq 1 \; avec NCO_i = \sum_{j=1}^{J} o_i(c_j) \tag{9}$$

NCO_i correspond au nombre de catégories inclues dans une commande o_i. Ainsi, le nombre maximum de splits possible lorsque toutes les catégories sont affectées à différents dépôts est égale à $NCO_i - 1$. $C(NCO_i, 2)$ correspond au nombre maximum possible de paire de catégories dans une commande. Si l'on prend 6 catégories, on a $NCO_i = 5$ et $C(NCO_i, 2) = 10$, ce qui nous donne un coefficient $\theta = 0.5$. En incluant θ_i, on ajuste alors le calcul des liens et qui nous donnera alors une meilleure estimation du nombre total de splits.

À partir de cette matrice $L(c_j, c_q)$, on va maintenant pouvoir estimer le nombre total d'out-links entre plusieurs dépôts. Cette estimation nécessitera trois différentes étapes. On commence d'abord par estimer le nombre de out-links entre une catégorie c_j d'un depôt w_k et une catégorie c_q dans un depôt w_g. On precise que $C(w_k)$ représente l'ensemble des catégories affectée à un depôt w_k. L'expression est la suivante :

$$outlink(c_j, c_q, w_k, w_g) = L(c_j, c_q) x_{jk} x_{qg}, \forall c_q, c_q \in C(w_k), j \neq q \tag{10}$$

On calcule ensuite l'ensemble des out-links liés à la catégorie c_j entre les dépôts w_k et w_g, qu'on exprime :

$$OUTL(c_j, w_k, w_g) = \sum_{q=j+1}^{J} outlink(c_j, c_q, w_k, w_g), \forall q \neq j, q \in C(w_g) \tag{11}$$

Vient ensuite le calcul des out-links entre deux dépôts donnés w_k et w_g exprimé par cette relation :

$$OUTL(w_k, w_g) = \sum_{j=1}^{J} \sum_{q=j+1}^{J} OUTL(c_j, w_k, w_g), \forall j \neq q, j \in C(w_k), q \in C(w_g) \tag{12}$$

Toutes ces étapes nous permettent alors d'exprimer la somme des out-links entre l'ensemble des dépôts que l'on définit comme suit :

$$LW = \sum_{k=1}^{K-1} \sum_{g=k+1}^{K} OUTL(w_k, w_g) \tag{13}$$

Notre objectif est de minimiser LW que l'on formule ainsi :

$$\text{Minimize } LW = \sum_{k=1}^{K-1} \sum_{g=k+1}^{K} OUTL(w_k, w_g) \tag{14}$$

LW est une borne supérieure du nombre réel des splits orders, $Min \sum_{k=1}^{K} y_{ik} - 1 \leq Min\ LW$. Cependant, cette approximation par l'intégration du coefficient θ_i dans le calcul du nombre d'outlinks est tout de même proche ou même égale dans certains cas au nombre réel de splits, et reste très pertinente à utiliser. On peut prendre l'exemple simple d'une commande composée de 4 catégories et 2 dépôts. On affecte 2 catégories à chaque dépôt. Ainsi, le nombre réel de splits vaut 1, le nombre d'outlinks vaut 4 et la somme pondérée des outlinks avec $\theta = 0.5$ vaut 2. On a donc bien une meilleure estimation du nombre réel de splits en intégrant une theta modification au nombre d'outlinks.

3.2.2 Description

La définition du concept de liens pour mesurer les similarités, relations entre les catégories impliqué dans la décision d'affectation des catégories dans les dépôts nous mène à considérer notre problème comme un problème de category clustering. La méthode de clustering est d'affecter des objets à des clusters en fonction de critères de similarités. Le principe de l'heuristique est donc d'affecter J categories, les objets, à K entrepôts, les clusters, en fonction de leurs liens définis à partir de I commandes clients. Nous avons formulé dans les précédentes parties le modèle mathématique de l'heuristique. Il s'agit maintenant de décrire le fonctionnement de l'heuristique nommée K-links clustering algorithm (KLCA). La méthode de résolution du KLCA est similaire à celle de l'algorithme du plus proche voisin.

On génère dans un premier temps une distribution aléatoire des J catégories dans les K entrepôts que l'on affecte à une variable CW_{trial} et on calcule la somme des outlinks entre les entrepôts LW_{trial} associée à cette distribution.
On cherche ensuite à réduire LW en modifiant la distribution des catégories à chaque itération. Pour cela, on peut appliquer deux stratégies. On déplace une catégorie à la fois d'un entrepôt à un autre ou encore on intervertit deux catégories affectées à deux différents entrepôts. Avant d'effectuer tout déplacement, on s'assure que celui-ci engendre bien la réduction de la valeur des outlinks tout en respectant les contraintes de capacités des entrepôts. Pour cela, on sélectionne aléatoirement une catégorie, puis on cherche les déplacement qui vont maximiser la réduction de LW en effectuant à chaque fois les deux stratégies de déplacements. Pour évaluer l'impact d'une relocalisation, on definit une variable $DL(c_j, w_k, w_g) = OUTL(c_j, w_g) - INL(c_j, w_k)$. Elle va nous permettre d'évaluer la différence entre la somme des outlinks avant et après la relocalisation. Concernant l'échange de deux catégories de deux entrepôts, on introduit la variable $Payoff(c_j, c_q, w_k, w_g) = DL(c_j, w_k, w_g) + DL(c_q, w_g, w_k) - 2L(c_j, c_q)$. À chaque nouvelle affectation, la valeur de LW ainsi que la distribution des catégories changent. Il est donc important d'actualiser à la fin de chaque itération la nouvelle distribution des catégories pour continuer d'améliorer notre solution. Il est cependant possible qu'aucune modification ne soit faite pour une itération donnée si aucun mouvement n'améliore LW. Nous avons définis le nombre d'itérations à 10 si et seulement si à chacune d'elle nous avons trouver un mouvement de améliorant, sinon nous sortons de la bouble avant.

La solution obtenue depend cependant de la distribution initiale des catégories générée aléatoirement. Dans certains cas, la solution peut converger vers un optimum local. Pour contrer ce problème, on effectue alors cette procédure sur différentes distributions. Une distribution sera considérée comme un essai, *trial*. Au bout d'un certain nombre d'essais, on sélectionne la meilleure solution considérée comme un optimum global. Il est en revanche nécessaire de fixer des conditions d'arrêts sur le nombre

d'essais effectués. Une des conditions d'arrêt est le nombre maximum d'essais possible appelé $Trial_{max}$, on continue à faire tourner notre algorithme tant que $trial \leq Trial_{max}$. On fixe pour notre heuristique la limite de 100 essais. Nous considérons qu'au bout de 100 essais, l'heuristique est capable de trouver une solution proche de l'optimale. Il est même possible qu'elle trouve une solution optimale bien avant les 100 essais. Il devient alors inutile de continuer à chercher une meilleure distribution des catégories. À chaque essai, si une nouvelle solution optimale est trouvée, on incrémente une variable $BestTrial$. Il devient donc facile de connaitre le nombre d'essais sans amélioration de la solution. On traduit cela par : $trial - BestTrial$ avec $trial$ le nombre d'essais jusque-là effectué. On introduit ensuite la variable $Trial^{non-improved}_{bound}$ correspondant au nombre maximum d'essais sans amélioration de la solution toléré. On fixe pour notre heuristique $Trial^{non-improved}_{bound} = 30$. Notre seconde condition d'arrêt sera alors, $(trial - BestTrial) > Trial^{non-improved}_{bound}$. Dès lors que la solution n'est pas améliorée sur 30 essais, on considère que la meilleure distribution de catégories trouvée est la solution optimale minimisant le somme des outlinks entre les dépôts et donc minimise le nombre de Splitting orders qui est notre objectif principal.

3.2.3 Formulation du modèle mathématique

Afin de pouvoir évaluer la performance de l'heuristique et la qualité de ses résultats, nous devons également modéliser ce nouveau modèle sur CPLEX. Nous avons vu qu'il était possible de voir le problème de deux façons, minimiser les outlinks ou maximiser les inlinks. Nous avons donc traité les deux cas sur CPLEX. Nous avons dans un premier temps minimiser les outlinks. Le modèle se définit de la sorte :

$$
\begin{aligned}
\textbf{Minimize } LW = & \sum_{k=1}^{K-1} \sum_{g=k+1}^{K} \sum_{j=1}^{J-1} \sum_{q=j+1}^{J} L(c_j, c_q) x_{jk} x_{qg} \quad k \neq q \\
\text{subject to : } & \sum_{k=1}^{K} x_{jk} = 1, && \forall j \\
& \sum_{j=1}^{J} x_{jk} \leq \mathrm{N}max, && \forall k \\
& \sum_{j=1}^{J} x_{jk} \geq Nmin, && \forall k \\
& x_{jk} \in \{1, 0\}, && \forall j, k
\end{aligned}
\tag{15}
$$

1. `Fonction objectif :` Dans le but de minimiser le nombre réel de fractionnements de commandes, on cherche à minimiser la somme des outlinks entre l'ensemble des entrepôts générée par l'affectation des catégories. En effet, on multiplie la matrice d'association des catégories $L(c_j, c_q)$ par les variables x_{jk} et x_{qg}. Dès lors, qu'une catégorie c_j est stockée dans un entrepôt w_k et une catégorie c_q est stockée dans un autre entrepôt w_g, on multiplie par 1 le lien θ modifié entre les catégories c_j et c_q.

2. `Contrainte 1 :` Cette contrainte va nous permettre d'assurer l'affectation d'une catégorie à un seul et unique entrepôt.
3. `Contrainte 2 :` Contrôle la capacité de chaque entrepôt, chaque entrepôt doit avoir au maximum $Nmax$ catégories de produits.
4. `Contrainte 3` : Contrôle la capacité de chaque entrepôt, chaque entrepôt doit avoir au minimum $Nmin$ catégories de produits.
5. `Contrainte 4` : Contrôle si une catégorie de produit j est entreposé ou non dans un entrepôt k. C'est la variable binaire d'affectation catégorie-entrepôt.

Le second modèle que nous avons traité a pour objectif de maximiser les inlinks. Il est équivalent à la minisation des outlinks. Il nous servira de comparaison. Les contraintes restent les mêmes, il a uniquement fallu modifier la fonction objectif comme suit :

$$\textbf{Maximize } LW = \sum_{k=1}^{K} \sum_{j=1}^{J-1} \sum_{q=j+1}^{J} L(c_j, c_q) x_{jk} x_{qk} \tag{16}$$

1. `Fonction objectif :` Dans le but de minimiser le nombre réel de fractionnements de commandes, on cherche à maximiser la somme des inlinks chaque entrepôts générée par l'affectation des catégories. En effet, on multiplie la matrice d'association des catégories $L(c_j, c_q)$ par les variables x_{jk} et x_{qk}. Dès lors, que les catégories c_j et c_q sont stockées dans un même entrepôt w_k,on multiplie par 1 le lien θ modifié entre les catégories c_j et c_q.

Comme pour le premier modèle, une phase de pre-processing sur VBA a été nécessaire pour générer les données essentielles à la résolution du modèle, à savoir la matrice de relations $L(c_j, c_q)$ et les données nécessaires à son calcul tel que θ_i et o_{ij} la matrice d'affectation des catégories de chaque commande.

3.3 K-Links Clustering Heuristique

3.3.1 Algorithme

Dans cette section, nous avons décidé dans un premier temps de présenter un flow chart de l'heuristique afin d'avoir une vue globale sur son fonctionnement. D'autre part, nous avons choisi d'intégrer des schémas explicatifs des deux opérateurs que nous avons utilisés : le relocate move (Selectionner une catégorie déjà affectée à un entrepôt puis la déplacer dans un nouvel entrepôt) et le exchange move (Selectionner deux catégories affectées à deux entrepôts différents et les échanger). L'objectif ici est de résumer ce qui a été dit dans la section 3.2.2. et de montrer l'impact qu'ont les deux opérateurs sur notre solution initiale.

Voici le logigramme de l'heuristique KCLA :

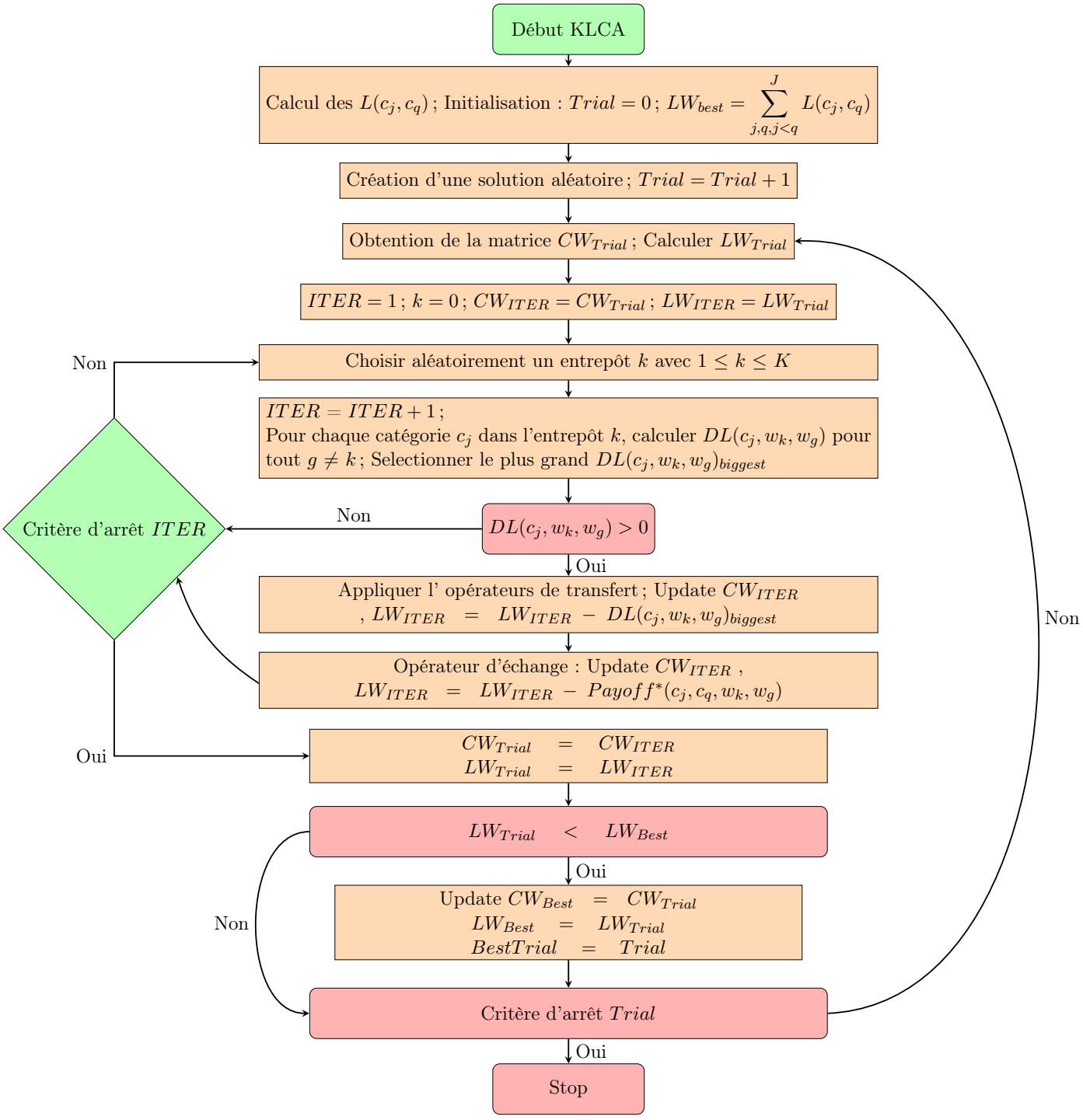

FIGURE 3 – Flowchart de l'algorithme K-links clustering

3.3.2 Opérateurs

On peut retrouver ci-dessous un schéma explicatif de l'impact de l'opérateur "relocate" sur nos liens :

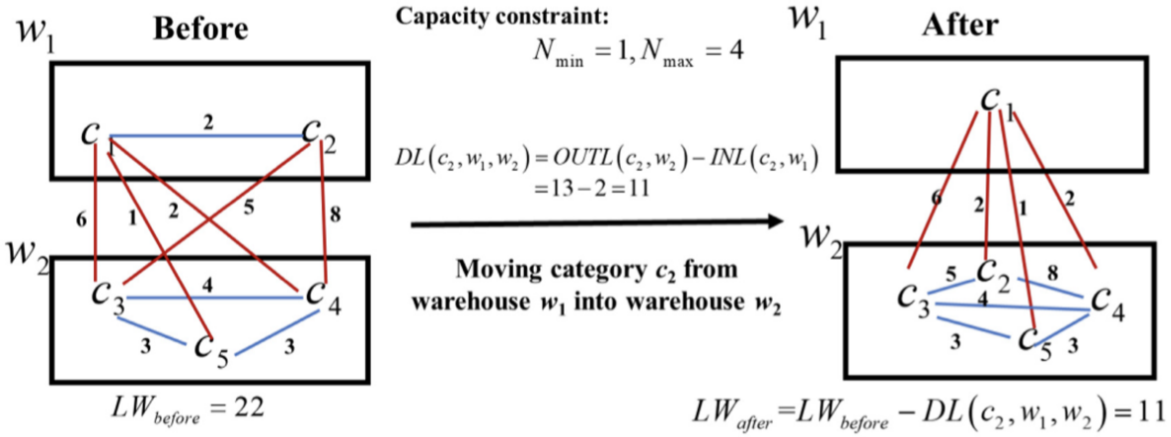

FIGURE 4 – Impact de l'opérateur 'relocate' sur les links

Dans cette figure, les inlink sont representés par des lignes bleues et les outlink par des lignes rouges. Comme on peut le voir ici, le changement d'affectation de la catégorie C_2 de l'entrepôt W_1 à l'entrepôt W_2 sous contraintes de capacité permet de supprimer un outlink et de le transformer en inlink. En effet, on recense dans la situaiton intiale 5 outlink et 4 inlink contre 4 outlink et 5 inlink dans la situation finale. Lorsqu'on somme les outlink, on obtient $OUTL(c_2, w_2) = L(c_2, c_3) + L(c_2, c_4) = 13$ dans la situation initiale. On obtient $INL(c_2, w_1) = L(c_2, c_1) = 2$ pour les inlinks. Quand on fait la différence, on retrouve $DL(c_2, w_1, w_2) = OUTL(c_2, w_2) - INL(c_2, w_1) = 13 - 2 = 11$ qui est la plus grande parmis les mouvements possibles. Lorsqu'on calcule ensuite LW on remarque qu'après application de l'opérateur relocate, on obtient $LW_{after} = LW_{before} - DL(c_2, w_1, w_2) = 22 - 11 = 11$, on a donc bien minimisé notre solution initiale de la somme des outlink égale à 22 en obtenant 11 pour la solution finale.

On retrouve ci-dessous le deuxième opérateur utilisé : 'Exchange move'

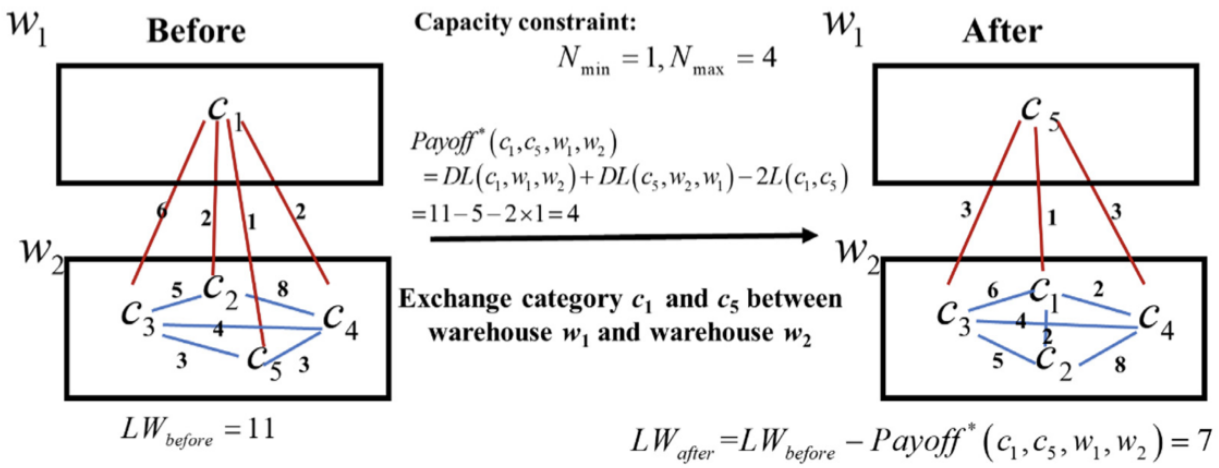

FIGURE 5 – Impact de l'opérateur 'Exchange' sur les links

Ici on reprend la solution précédemment obtenue à l'aide de l'opérateur relocate et on cherche à determiner quelles catégories échanger dans le but de minimiser encore une fois la sommes des outlinks. Pour cela on utilise la formule $Payoff(c_j, c_q, w_k, w_g) = DL(c_j, w_k, w_g) + DL(c_q, w_g, w_k) - 2L(c_j, c_q)$ et on choisit la paire de catégories ayant la plus grande valeur de $Payoff$. Ici c'est un échange entre les catégories 1 et 5 qui donne la plus grande valeur de $Payoff$ égale 4. Lorsqu'on échange ces deux catégories, on calcule la solution finale de la somme des outlinks comme suit : $LW_{after} = LWbefore - Payoff^*(c_1, c_5, w_1, w_2)$ et obtient $LW_{after} = 7$. Ainsi grâce à l'opérateur exchange, nous avons de nouveau pu minimiser la somme des outlinks.

4 Résultats

Dans cette section, nous allons présenter les différents résultats obtenus grâce aux modèles mathématiques précedemment presentés. Nous allons dans un premier temps exposer les resultats des CPLEX et de l'heuristique puis dans une seconde partie nous allons comparer les deux méthodes et determiner laquelle est la plus performante.

4.1 Présentation

Pour la partie qui suit, nous allons afficher les résultats de nos différents modèles mathématiques obtenus via CPLEX et l'heuristique. Tous nos test ont été effectués sur un macbook pro de 2018 avec 8 Go de RAM et un processeur Intel Core i5 quadcore cadencé à 2.3 GHz. Pour la partie programmation de l'heuristique, nous avons choisi le language Python pour deux raisons : d'une part car c'est un language montant que nous souhaitions découvrir et apprendre. D'autre part car c'est un language riche en librairies et modules en tout genre permettant de simplifier de nombreuses étapes allant de la lecture de données aux calculs mathématiques en tout genre (notamment les calculs matriciels avec la librairie numpy) mais également de proposer des visualisations claires et agréables.

Nous avons deux dataset à notre disposition recupérés sur Kaggle et légèrement modifiés dont les caractéristiques sont resumées dans le tableau ci-dessous :

Paramètres	Dataset complet	Dataset raccourci
Nombre de commandes I	22 061	117
Nombre de catégories J	60	15
Nombre d'entrepôts K	10	3
Capacité minimale N_{Min}	2	2
Capacité maximale N_{Max}	15	10

Nous avons choisi de tester nos différents modèles sur le dataset raccourci, d'une part pour des raisons de performances, d'autre part car CPLEX est tout bonnement incapable d'afficher une solution avec le dataset complet, même après plus de 4h de compilation du fait de la NP-complétude du problème et du dataset aussi conséquent.
Dans les parties qui vont suivre nous allons présenter les différents résultats obtenus.

4.1.1 Modèle optimal

Dans le cas du modèle optimal, on décide de minimiser directement le nombre de splits décrit par la formulation : $Minimize \sum_{i=1}^{I}(\sum_{k=1}^{K} y_{ik} - 1)$. On fixe pour la résolution de ce modèle 3 entrepôts avec chacun une capacité minimale de stockage de 2 catégories de produits et une capacité maximale de 10. On va utiliser CPLEX et on obtient les résultats ci-dessous :

```
--------------------------------------------------------------------------------

Number of Splits = 141

Number of inlink = 368.05979022

Number of outlink = 381.9402098

--------------------------------------------------------------------------------

With 3 warehouses of capacity Nmin = 2 and Nmax = 10

Warehouse  1  should store the categories : { 8,  10, }

Warehouse  2  should store the categories : { 2,  9,  13, }

Warehouse  3  should store the categories : { 1,  3,  4,  5,  6,  7,  11,  12,  14,  15, }


--------------------------------------------------------------------------------

solving time ~= 21.661572021 s
--------------------------------------------------------------------------------
```

FIGURE 6 – Résultats du modèle optimal obtenus via CPLEX

Ainsi avec ce premier modèle on obtient 368.05 inlink et 381.94 outlink. Pour notre dataset de 117 orders pouvant contenir jusqu'à 15 catégories différentes on pourra ainsi expédier toutes nos commandes avec 141 splits seulement.
On constate, du reste, qu'en simulant toutes les combinaisons possibles, le solveur a tendance à remplir un entrepôt tout en gardant les deux autres autour de leurs capacités minimales. Concernant l'interpretation de ces résultats, rappelons nous que la somme des y_{ik} sur tous les entrepôts k correspond au nombre de dépôts utilisés pour satisfaire une commande i contenant j catégories. On peut alors supposer que les catégories c_8, c_10 sont contenues dans beaucoups de commandes communes mais beaucoup moins liées aux autres catégories et sont donc affectées au même entrepôt (ici w_1) par le solveur, dans le but de limiter les splits. Ici, les contraintes de capacités fixées en amont sont assez souples pour notre dataset mais on peut aisément imaginer qu'en réduisant la capacité maximale des entrepôts le solveur compilera beaucoup plus longtemps avant d'afficher une solution qui du reste sera nettement degradée. Ce constat est également applicables aux autres paramètres tels que le nombre d'entrepôts et trouve son origine à la NP-complétude du problème. Pour proposer un résultat satisfaisant en un temps d'execution raisonnable nous avons alors testé notre modèle heuristique.

4.1.2 Modèle heuristique

Pour ce modèle, rappelons que le modèle mathématique initial a été transformé. Nous considérons maintenant des clusters de catégories construit à partir de liens ou 'link' en anglais. Quand deux catégories sont incluses dans une même commandes, leur lien vaut 1, 0 sinon, puis une théta-modification est appliquée comme expliqué dans la section 3.2.1. Si ces catégories sont stockées dans le

même entrepôt on parle d'inlink, un lien qui ne causera pas de splitting order sinon on parle d'outlink. L'interet du problème revient alors à minimiser le nombre d'outlink ou à maximiser le nombre d'inlink.

Voici les résultats obtenus par CPLEX avec ce modèle :

```
-----------------------------------------------------------------------------------------------

Number of inlink = 381.19558779

Number of outlink = 368.80441223

-----------------------------------------------------------------------------------------------

With 3 warehouses of capacity Nmin = 2 and Nmax = 10

Warehouse  1  should store the categories : { 1,  2,  3,  5,  7,  9,  12,  13,  14,  15, }

Warehouse  2  should store the categories : { 6,  8,  10, }

Warehouse  3  should store the categories : { 4,  11, }


-----------------------------------------------------------------------------------------------

solving time ~= 4.172843018 s
-----------------------------------------------------------------------------------------------
```

FIGURE 7 – Modèle heuristique : maximisation des inlink

```
-----------------------------------------------------------------------------------------------

Number of outlink = 368.80441223

Number of inlink = 381.19558779

-----------------------------------------------------------------------------------------------

With 3 warehouses of capacity Nmin = 2 and Nmax = 10

Warehouse  1  should store the categories : { 6,  8,  10, }

Warehouse  2  should store the categories : { 4,  11, }

Warehouse  3  should store the categories : { 1,  2,  3,  5,  7,  9,  12,  13,  14,  15, }


-----------------------------------------------------------------------------------------------

solving time ~= 4.276228027 s
-----------------------------------------------------------------------------------------------
```

FIGURE 8 – Modèle heuristique : minimisation des outlink

On remarque dans un premier temps que les résultats sont les mêmes qu'on maximise les inlink ou qu'on minimise les outlink. Cela est logique puisque le modèle mathématique est le même pour les deux cas.

Ici on obtient 368.8 outlinks et 381.19 inlink. Le solveur va par ailleurs adopter le même comportement que pour le modèle optimal en remplissant au maximum un entrepôts tout en laissant les deux autres aux capacités minimales. Le raisonnement est analogue au premier modèle : En calculant les liens entre toutes les catégories, le solveur a affecté les 10 catégories les plus similaires dans le même entrepôt puis le reste dans les entrepôts restants. On obtient ainsi une solution assez similaire au premier modèle avec néanmoins quelques différences d'affectation de catégories comme pour c_2 qui est ici affectée à l'entrepôt le plus utilisé alors qu'elle était assignée à un des entrepôts vides dans le premier modèle.

La grande différence qu'on constate ici est le temps de compilation. En effet, en utilisant le même outils de résolution, CPLEX on obtient un temps de résolution de 4.27 secondes seulement pour le modèle heuristique contre 21.66 secondes pour le modèle optimal tout en conservant une solution correcte. On comprend alors l'intêret du modèle heuristique ici notamment lorsqu'on va utiliser un dataset plus conséquent ou des contraintes de capacités et d'entrepôts plus limitantes : éviter un temps de compilation qui explose sans énormement dégrader la solution. Par ailleurs, nous aurions pu détourner une nouvelle fois le problème et étudier le dual lagrangien du modèle et non le primal. Le but étant d'obtenir les shadow cost via une analyse de sensibilité et donc d'avoir implicitement les contraintes de capacités N_{min} et N_{max} les plus adaptées notre problème permettant de dimensionner nos entrepôts de manière à avoir un système d'execution de commande performant. Dans la vie réelle ces dimensions sont stratégiques et sont assez pertinentes à étudier.

4.1.3 Heuristique

Une fois nos deux modèle testés sur CPLEX, dans une optique de gain de performance et de temps d'execution, il est assez courant de programmer le modèle heuristique à l'aide d'un langage informatique. Très souvent on choisit le C++ car actuellement c'est le langage le plus performant qui existe. Néanmoins, faute de connaissance et de temps, nous avons décidé d'utiliser Python pour programmer notre heuristique. Il faut alors savoir que les résultats obtenus ici en terme de temps d'execution auraient été presque 100 fois meilleurs avec C++
La solution finale obtenue par l'heuristique correspond ici à une minimisation de la sommes des outlink. Ayant prouvé précédemment que le problème était identique qu'on considère les inlink ou les outlink, nous avons décidé de travailler uniquement sur les outlink pour cette partie.

On obtient ainsi le résultat suivant :

```
executed in 2.33s, finished 21:18:38 2020-12-20
------------------------------------------------------------------------

Number of inlink = 381.195587745588

Number of outlink = 368.80441225441217

------------------------------------------------------------------------

 Warehouse  1  should store the categories :   { 4, 11, }

 Warehouse  2  should store the categories :   { 6, 8, 10, }

 Warehouse  3  should store the categories :   { 1, 2, 3, 5, 7, 9, 12, 13, 14, 15, }

------------------------------------------------------------------------
```

FIGURE 9 – Solution finale de l'heuristique KLCA - Python

On remarque que la solution est identique à celle obtenue par CPLEX en 2.33 secondes. on peut le visualiser de manière plus claire grâces à la librairies matplotlib en affichant le remplissage des entrepôts :

FIGURE 10 – Répartition des catégories c_j dans les entrepôts w_k

On voit bien que comme avec CPLEX, ici on a un entrepôt dans lequel on va affecter un maximum de catégories jusqu'à saturer ce derniers et deux autres qui vont acceuillir les catégories restantes tout en respectant les contraintes minimales. On peut ici se poser une question : pourquoi obtient-on une solution similaire à CPLEX ?
Dans notre cas, la réponse à notre questions provient en réalité du dataset et surtout des paramètres que nous avons fixé. En effet, le dataset que nous avons choisi est de base assez peu conséquent et de surcroit nous avons fixé en entrée de chacune de modélisation des paramètres de capacités très souples. Ainsi, nous obtenons en réalité une solution optimale très rapidement comme le montre le graphe ci-dessous de la convergence de LW_{Best} :

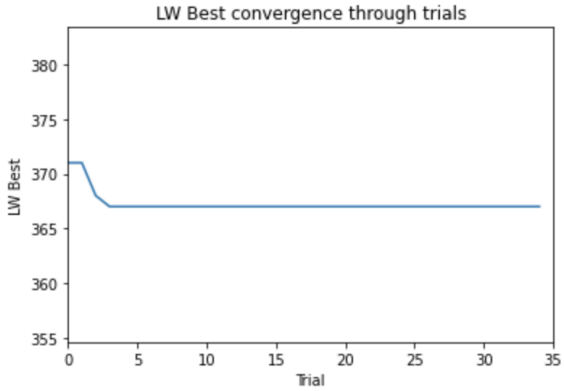

FIGURE 11 – Convergence de LW_{Best}

On voit clairement sur ce graphe que nous avons amélioré la solution deux fois seulement avant d'atteindre une très bonne solution, qui, dans notre cas, s'avère être optimale.
Nous avons alors décidé de modifier les paramètres d'entrée plusieurs fois et de consigner nos résultats pour comparer l'heuristique KLCA et le modèle CPLEX de manière plus rigoureuse. Dans la partie qui suit nous allons exposer les différents résultats obtenus ainsi que notre analyse.

4.2 Analyse numérique

Afin de comparer les deux méthodes de résolution nous avons fait des analyses de résultat en fonction du nombre maximal $Nmax$ de produits pouvant être stockés dans un même entrepôt. Nous comparons le temps de résolution et le résultat obtenu par chaque méthode. Notons que pour la suite de cette section nous considérons les résultats liés à la mesure des outlinks.

Pour cela, nous utilisons pour les deux méthodes les mêmes données et paramètres provenant du dataset décris précédemment de : $I = 117$, $J = 15$, $K = 3$ et $Nmin = 2$

Nous calculons le Gap en poucentage, entre ces deux méthodes de la manière suivante :

$$Gap\% = \frac{(KLCA - Cplex)}{(Cplex)}$$

$Nmax$	Cplex	K-LCA	Gap%	Cplex TIME (en s)	K-LCA TIME (en s)
11	313.13689	313.17592	0,0124	2.8	0.98
10	368.80441	368.80441	0,0000	7.93	1.22
9	412.20847	412.20847	0,0000	15.83	0.95
8	446.10991	446.12085	0,0024	15.56	1.13
7	462.97561	463.73389	0,1637	23.42	1.28
6	494.45473	494.45473	0,0000	27.89	0.78
5	518.89280	518.89280	0,0000	18.77	0.92

TABLE 1 – Evolution du temps de résolution et du résultat optimal en fonction de $Nmax$ pour $K = 3$.

Pour : $K = 4$ et $Nmin = 2$

$Nmax$	Cplex	K-LCA	Gap%	Cplex TIME (en s)	K-LCA TIME (en s)
9	441.73265	441.73265	0,0000	44.04	0.78
8	487.21856	487.54104	0,0661	202.22	0.91
7	518.59467	518.79200	0,0380	221.77	1.37
6	535.36944	535.36944	0,0000	282.13	1.79
5	558.62504	558.62504	0,0000	367.44	1.92
4	583.92982	584.01020	0,0137	233.44	3.34

TABLE 2 – Evolution du temps de résolution et du résultat optimal en fonction de $Nmax$ pour $K = 4$.

On constate que les résultats obtenus par l'heuristique sont de très bonne qualité. Dans l'écrasante majorité des cas l'heuristique trouve la solution optimale au problème transformé. Losque ce n'est pas le cas, les solutions trouvées demeurrent très bonnes. En effet, les GAP sont tous inférieurs à 0,1%. De plus, nous observons clairement que l'heuristique est plus efficace que le CPLEX sur le critère du temps d'exécution car l'heuristique est en moyenne 16 fois plus rapide que le modèle CPLEX pour $K = 3$. Cela se confirme pour $K = 4$: CPLEX met en moyenne 145 fois plus de temps que l'heuristique pour afficher une solution.

4.3 Dataset large

Il est à noter que, le grand avantage que présente l'heuristique face à la programmation linéaire quant au critère du temps d'exécution importe peu en l'état. En effet, la décision qui consiste à attribuer des catégories de produit à des entrepôts est au minimum tactique, l'horizon de décision se discutant en mois. Ainsi, dans la configuration actuelle des choses, qu'un programme nécessite 1 seconde ou 300 secondes pour trouver une solution a peu d'importance, hormis si l'on considère "l'expérience utilisateur", qui, dans ce type de problème, nous importe peu. Toutefois à l'heure ou le buzzword "Big Data" prend une ampleur considérable, nous ne sommes pas sans savoir que, sur le terrain, dans la vraie vie, les jeux de données, notamment dans le e-commerce sont bien souvent très larges. On gère parfois des données dont la taille se mesure en pétaoctets.

Naturellement, nous souhaitions souligner le réel aspect avantageux de l'heuristique. Le dataset raccourci sur lequel tous ces tests ont été effectués est à des lieux de représenter la réalité. Il n'a été choisi que pour sa capacité à etre exploité sur CPLEX. Toutefois, le dataset complet présenté dans la section 4.1 s'avère être bien plus en adéquation avec les données des grands groupes de e-commerce (pour à peine une semaine de commande). C'est avec plaisir que nous mentionnons que notre code (en python) nous fourni des solutions, dont on estimera la qualité par analogie comme étant bonne, en un temps moyen d'environ 3 minutes, là ou il faudrait plus que probablement des années à un solveur pour trouver la solution optimale. Rappelons le, pour des décisions d'horizon tactique, 3 minutes est un temps plus qu'acceptable, et c'est ce réel aspect du temps d'exécution qui fait de l'heuristique une solution exploitable et viable.

5 Conclusion

5.1 Amélioration de l'heuristique

Nous venons de souligner que notre dataset était de petite taille, ainsi l'heuristique parvient à trouver des solutions d'extrême qualité. Nous souhaitons mentionner ici des prémices d'idées d'améliorations, ou tout bonnement de différentiation, à titre de suggestion pour des travaux ultérieurs.

La perturbation de nos solutions et ainsi la minimisation des risques d'une convergence vers un minimum local est dans notre cas assurée par la structure de l'algorithme en lui même. En effet, dans chaque *trial* la solution initiale est totalement aléatoire. Toutefois, pour permettre plus de diversité dans les solutions initiales (utile pour les gros datasets) il est intéressant de proposer d'autres types de création de solution initiales avant tout mouvement de recherche local. On pense par exemple à des heuristiques couplées à des facteurs aléatoires pour promouvoir la diversité des solutions. Dans notre cas une idée serait de randomizer les heuristiques de k - $Nearest\ Neighbour$ avec initialement les k catégories les moins dépendantes ou encore de $Clark\ \&\ Wright$ une fois les N_{Min} remplis dans chaque entrepôts.

Nous pensons aussi que si la taille des données est grande, les statistiques peuvent être un outil précieux dans la performance de l'heuristique. Toutefois, pour ne pas provoquer de convergence prématurée nous suggérons de réaliser cela après un certains temps de sorte à ce que le "balancing" opère et que les minimums locaux ne prédominent pas. En effet, à l'image des phases d'intensification des méta-heuristiques nous pensons qu'il est interessant de reconnaitre des parterns récurrents sur les liens et leurs présence dans les solutions en tant que 'outlink' ou 'inlink' ou encore sur les association 'catégorie-entrepôts' les plus récurrentes. Ces patterns peuvent aider le modèle à construire des solutions bonnes initialement et simplifier ainsi la recherche locale. Nous pensons aussi par exemple qu'il est possible de commencer par faire des "paires" de catégories qui maximisent les inlink dans ces paires pour ainsi ensuite résoudre un problème dont la taille a été divisée par deux. Ce qui en sommes est déjà réaliser dans cet article qui s'intéresse directement à des catégories de produits puisqu'à l'interieur de celles-ci les SKU sont déjà clusterisées de manière réfléchie.

Toutes ces solutions que nous avons imaginées sont en sommes proposées dans le cas d'une donnée trop large comme étant parfois améliorantes de la solution finale ou tout bonnement améliorante du temps d'execution. Il existe par exemple des problèmes opérationnel très complexe dont l'enjeu est non pas de trouver une solution optimale mais de trouver une solution faisable, à l'instar de ces problèmes nous pensons qu'il peut être envisageable de réduire légèrement les performances d'une heuristiques si cela permet de traiter un jeu de donnée plus large qui sur le long termes reflètera la réalité avec plus de précision.

5.2 Ouverture

La littérature scientifique en termes d'optimisation des processus liés aux supply chain est riche. Toutefois, l'application dans les entreprises est encore en retrait par rapport à l'état de l'art, souvent faute de moyen financiers ou techniques. Cependant, le monde de demain, ne cessera d'être demandeur d'écologie et d'optimisation. Dès lors, être maître de sa donnée, et par exemple diminuer son émission de CO2 ou de déchets liées au packaging seront, pour sur, des gap statégiques pour les entreprises. Nous pensons donc qu'un tel travail saura sans aucnun doute trouver dans la recherche des volontaires pour en améliorer des aspects, mais la migration large de telles techniques dans le monde de l'industrie, si l'on oublie les grands groupes, serait une toute autre ouverture tout aussi prometteuse. Ainsi nous pourrions considérer l'ouverture d'un tel travail comme un effort de normalisation des ces méthodes voir même le développement de progiciel compatbile avec les ERP les plus répandus afin de proposer à l'industrie une façon simple de travailler sur son impact sur les enjeux de demain.

Références

[1] Acimovic, J. A. (2012). *Lowering outbound shipping costs in an online retail environ- ment by making better fulfillment and replenishment decisions.* Cambridge, MA : Massachusetts Institute of Technology.

[2] Acimovic, J., Graves, S. C. (2014). *Making better fulfillment decisions on the fly in an online retail environment.* SSRN Electronic Journal. https ://doi.org/10.2139/ ssrn.2166687.

[3] Acimovic, J., Graves, S. C. (2017). *Mitigating spillover in online retailing via replenishment.* Manufacturing Service Operations Management, 17(1), 34–51. https ://-doi.org/10.1287/msom.2014.0505.

[4] Catalán, A., Fisher, M. (2012). *Assortment allocation to distribution centers to minimize split customer orders.* Manufacturing Service Operations Management, 19(3), 419–436. https ://-doi.org/10.1287/msom.2016.0614.

[5] Site internet librairie Python Numpy *https ://numpy.org*

[6] Site internet librairie Python Pandas *https ://pandas.pydata.org*

[7] Site internet librairie Python Matplotlib *https ://matplotlib.org*

[8] Site internet Notebook Python Jupyter *https ://jupyter.org*

[9] Site internet de Data Science *https ://towardsdatascience.com*

[10] Site internet d'aide au développement LaTex *https ://fr.overleaf.com*

[11] Donald E. Knuth (1986) *The TEX* Book, Addison-Wesley Professional.