RECHERCHE TRANSPORT OPTIMAL

On a pensé à prendre un algorithme de coût minimum où il y aurait 2 sous-ensembles dans un ensemble de points : un sous-ensemble de points comprenant les entrepôts et un autre sous-ensemble de points comprenant les magasins.

**Données :**

* i : un entrepôt où i un sommet appartenant à {1,2,…,n}
* j : un magasin où j un sommet appartenant à {1,2,…,m}
* u : un livreur où u appartenant à {0,1,2,…,k}
* I : la quantité d’entrepôt maximum
* J : la quantité de magasin maximum
* U : la quantité de livreur maximum
* P : le prix de l’essence
* T : le temps qu’un livreur doit passer dans une journée de travail
* tju : le temps passé au magasin par le livreur u
* tju : le temps passé en entrepôt par le livreur u
* Pu : le coût en euros de payer un livreur par heure
* Pi : le coût en euros d’acheter un entrepôt
* Qu : la quantité de marchandise qu’un livreur peut transporter
* Qj : la quantité de marchandise qu’a besoin le magasin
* Qi : la quantité de marchandise que possède un entrepôt
* Dij : la distance entre i et j avec (i,j) en km
* Cij : le coût d’une arête allant de i à j avec (i,j)
* Tiju : le temps d’une arête allant de i à j par livreur u avec (i,j)

**Variable(s) :**

* Xiju : Vaut 1 si le livreur u passe par l’arête (i,j) et 0 sinon
* Xi : Prendre l’entrepôt i
* Xj : Aller au magasin j

**Objectif(s) :**

* L’objectif est de minimiser le coût Cij des arêtes entre les sommets (entrepôts et magasins), le coût d’achat d’entrepôt minimisant ainsi la quantité d’entrepôt, le coût de paye d’un livreur par jour en heure
* ∑(Xiju × Cij) + ∑(Xi × Pi) + ((∑tiu+∑tju) × Pu)

**Contraintes :**

* La somme des Xj doit être égale à J
* I est inférieur à une quantité d’entrepôt max que l’on veut prendre
* U est inférieur à une quantité max de livreurs dans chaque zone géographique
* T est inférieur à 7H de travail par jour
* (Le prix de l’essence P fois la distance Dij fois le nombre de fois que l’on prend l’arête Dij) ne doit pas être trop élevé
* La quantité Qu que peut transporter un livreur ne doit pas être dépassé
* La quantité Qj que veut le magasin doit être satisfaite
* La quantité Qi que possède un entrepôt ne doit pas (et ne peut pas) descendre en dessous de 0

Définition des besoins:

Le but est de déterminer quels sont les entrepôts avec qui nous avons besoin de travailler.

Nous devons prendre en compte :

-où sont les magasins à livrer ? (Coordonnées GPS)

-où sont les entrepôts disponibles ? (Coordonnées GPS)

-où habitent les livreurs ?

-quelle distance entre chaque magasin et chaque entrepôt ?

-combien dispose-t-on de livreurs ?

-Le coût de faire travailler un livreur ?

-Le coût de stockage d’un entrepôt pour une certaine quantité de marchandise.

-le nombre de magasin qu’un livreur peut faire en une journée de travail.

-le nombre de kms qu’un livreur peut faire par jour ?

-capacité de marchandise que le livreur peut transporter en une seule fois

**Idées :**

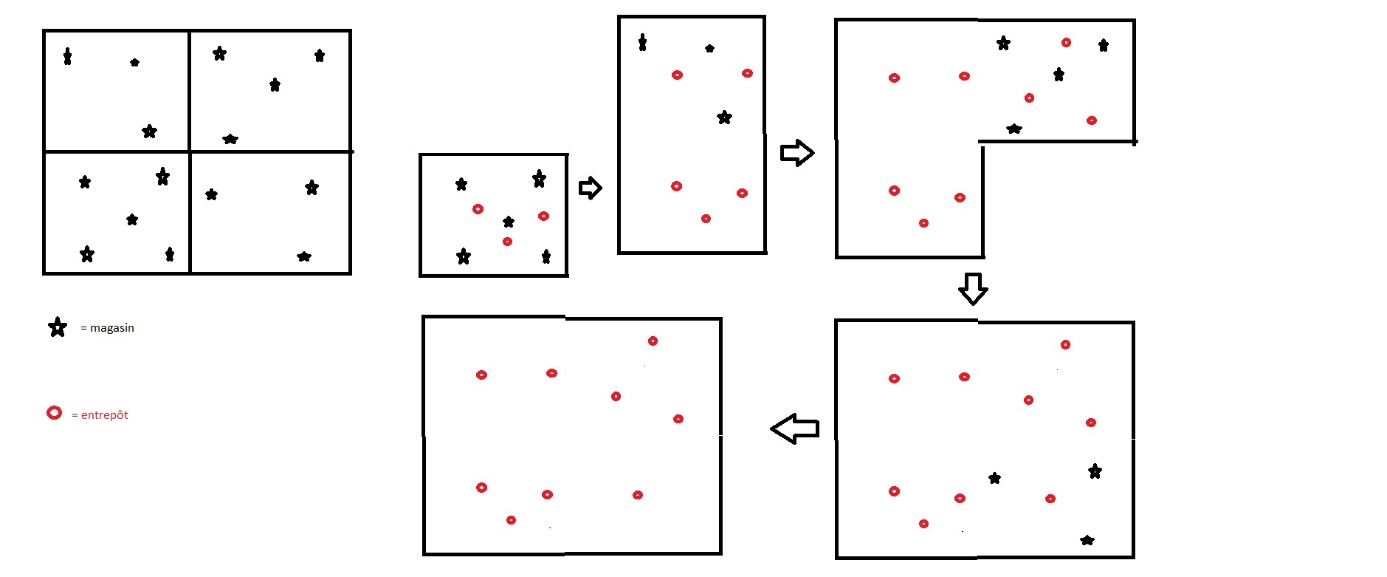
-programmation linéaire (avec aussi les solveurs excel)

Idée avec les graphes. Problème des flots. Arbres couvrants de poids minimum.

**Avancée recherche :**

Le deep learning se base sur des données non structurées tandis que le machine learning se base sur des données structurées. Les données structurées sont des données classées, nettoyées, organisées. Et c’est justement ce à quoi l’on aura à faire avec des données sur les entrepôts, les magasins, les livreurs etc…. En plus de cela, le deep learning sert la plupart du temps pour la reconnaissance et la création d’image (deep fake) donc non approprié au type de tâche que nous devons réaliser qui se base plutôt sur une décision optimale quant au coût de déplacement et livraison entre les entrepôts et les magasins. En conclusion, nous pensons que le machine learning serait plus approprié pour nous que le deep learning et donc que les réseaux de neurones artificiels qui constituent le deep learning ne sont plus intéressants.

Vu que les données (si on prenait la France entière) seraient trop grandes et trop volumineuses avec un trop grand nombre de possibilité, on a donc décidé de faire un découpage. On ne sait pas encore le type de découpage (dép, rég, iris etc….). L’idée que j’ai eu serait de faire un découpage comme mis sur la photo et il nous permettrait de savoir quel entrepôt nous devrons garder. Ensuite, on pourrait faire la même chose en fonction des livreurs



**Idée complémentaire :**

Nous pouvons aussi, au lieu de calculer avec un algorithme la position des meilleurs entrepôts à choisir (comme sur la photo), faire livreur par livreur. C'est-à-dire que l'on met en place le problème du voyageur de commerce en cherchant pour chaque livreur un par un, la façon la plus optimale, de passer par tous les magasins. On met en place l'algorithme pour le premier livreur, ensuite les magasins déjà livrés par le premier livreur s'enlèvent (dans notre programme) car déjà livrés. Puis on relance l'algorithme pour le deuxième livreur avec les magasins restants et ainsi de suite jusqu'à ne plus avoir de magasins à livrer. On rappelle plusieurs contraintes: tout d'abord le livreur doit passer en tout premier par un entrepôt (pour se charger), l'entrepôt étant celui le moins loin. Et ensuite on lance l'algorithme et le livreur passe par le plus de magasin possible en revenant au point de départ à la fin avec une limite d'heure de travail par exemple, ou de magasin à livrer. Puis on passe au deuxième livreur en supprimant les magasins par lesquels les anciens livreurs sont déjà passés. Cette méthode peut sans doute être plus longue car pour chaque livreur, on lance deux algorithmes: un premier pour calculer le plus court chemin entre le livreur et l'entrepôt. Et un deuxième pour calculer les magasins dans lesquels le livreur va passer. Tout ça fois le nombre de livreur. Et à la fin on se retrouve avec la quantité de livreur, et d'entrepôt dont l’on a besoin et l'ensemble des magasins qui ont été livrés.

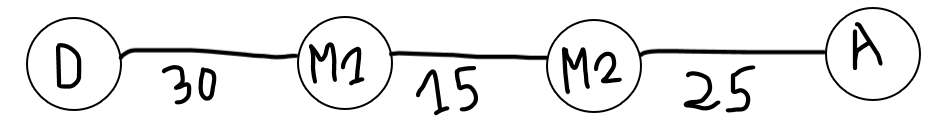
**Idée :**

Utiliser un algo du plus court chemin. Ou bien un arbre couvrant de poids minimum.

Il faut décomposer le problème en plusieurs étapes. La première, est de déterminer quels sont les entrepôts à utiliser. Il faut couvrir une zone suffisamment grande pour desservir tous les points d’intérêts (magasins), pour autant, il ne faut pas que 2 entrepôts soient trop proches, car il serait dommage de louer un entrepôt pour rien.

Une fois les entrepôts déterminés, il faut choisir le chemin le plus optimal pour la livraison dans les points d’intérêts. Pour choisir le trajet, il faut répondre à une contrainte : le temps de travail que peut effectuer un employé. Fixons ce seuil à 7 heures. Pour choisir le plus court chemin, il faut chercher en réalité le chemin le plus rapide (en temps) et non le plus court (en km). On va utiliser un graphe pondéré. Les sommets représenteront les points d’intérêt et les arrêtes représenteront les trajets entre deux points. Les arêtes seront pondérées par le temps de trajet entre deux points. Pour représenter le temps de travail effectué dans le magasin, on utilise deux sommets par magasins, étant reliés par une arrête, elle-même pondérée par le temps de travail. En voici un exemple illustratif :

Dans l’exemple ci-dessous, le sommet D représente un point de départ, le point A représente un point d’arrivée. Les points M1 et M2 représentent UN magasin. Quand le livreur arrive au magasin, il est au point M1 et quand il part du magasin, il part du point M2. La pondération de l’arrête entre M1 et M2, représente le temps de travail nécessaire sur le magasin. Les pondérations des arêtes (D-M1), (M2-A), représentent le temps de trajets entre ces points là.



**Idée :**

On a découvert le problème de tournées de véhicules qui ressemble presque à la même chose que notre problématique à nous. Le but est de minimiser le coût de livraison des biens tout en essayant de livrer une liste de clients. Dans ce problème, on ne peut pas trouver une solution optimale, seulement une solution de bonne qualité. Il fonctionne à l’aide contraintes avec la méthode de Branch and cut qui permet de diminuer le plus possible le nombre de possibilités et donc trouver une solution la plus optimale possible.

1. **Lecture algorithme pour problème de tournée de véhicules :**

Nous avons trouvé en cherchant, que le problème que l’on se pose à savoir passer par tous les magasins, en ayant plusieurs entrepôts et en ayant plusieurs contraintes comme des contraintes de temps, et bien a déjà été questionné, notamment par Clarke et Wright et bien d’autres.

Dans [ce PDF](http://perso.eleves.ens-rennes.fr/people/clement.legrand-lixon/rapport.pdf#:~:text=L%E2%80%99algorithme%20Clarke%20%26%20Wright%20%28CW%29%20est%20un%20algorithme,le%20nombre%20de%20v%C3%A9hicules%20disponibles%20n%E2%80%99est%20pas%20respect%C3%A9e%29.), on peut constater une description du problème avec plusieurs solutions, plusieurs algorithmes et plusieurs variantes de solutions. On va donc les décrire en prenant les dépôts comme nos entrepôts, les clients comme nos magasins et les véhicules comme nos livreurs :

* Véhicule Routing Problem (problème de tournées de véhicules -> VRP) : On a 1 dépôt pour plusieurs clients et on dispose de plusieurs véhicules et le but est de minimiser la longueur du réseau donc de minimiser le nombre de tournées c’est-à-dire le nombre de fois qu’un véhicule fais une boucle (part et revient au point de départ) en passant par un ensemble de magasins.
* VRP associé au CVRP : Dans cette variante, on ajoute au VRP une contrainte de capacité de véhicule avec une demande du client où le véhicule ne doit pas dépasser sa capacité maximale dans chaque tournée qu’il va réaliser. C’est-à-dire qu’une fois atteint cette limite, le véhicule doit revenir au dépôt.
* Algorithme de Clarke & Wright : Cet algorithme est composé de plusieurs étapes : une première où on va attribuer à chaque client un véhicule. Ensuite, un véhicule réalise une tournée. Chaque tournée dépend des *savings* et le nombre de clients livrés dépend donc aussi des *savings.* Ces *savings* sont : *s(i, j) = ci0 + c0j − λcij + µ|ci0 − c0j | +* avec i et j correspondant à des clients, c.. correspondant à la distance entre le client et le dépôt (dépôt = 0), d. correspondant à la demande du client. (ci0 + c0j) sont la somme des distances de i au dépôts et de j au dépôt. Le paramètre *λ* est appelé *route shape parameter*, le paramètre *µ* évalue l’asymétrie (la différence) entre la distance de i vers le dépôt et la distance de j vers le dépôt, le paramètre permet de s’intéresser aux clients qui ont la plus grosse demande en premier. Dans l’algorithme, on va mettre les *savings* à 0 dès l’instant où l’on choisit de fusionner i et j et donc que le véhicule n’aille pas que à i mais aussi à j. Et donc, à force de mettre les *savings* à 0, on va arriver à la fin à un *savings* total de tous les clients de 0 et c’est à ce moment que l’algorithme s’arrêtera. (Voir image)
* Heuristique de Arnold & Sörensen : Cette heuristique commence par réaliser une solution initiale via l‘algorithme de CW. Ensuite, pour chaque arête, on la considère comme étant la pire arête qui a pu être choisi dans tout le graphe et on décide de parcourir tout le graphe dans sa totalité puis après avoir tout parcouru, de choisir le voisin de l’arête qui est le meilleur à garder. La boucle de l’algorithme s’arrête au bout de 3 minutes pour toutes les arêtes. Maintenant, on va détailler le calcul de la pire arête et comment on choisit quel voisin est le mieux. Pour cela il existe 3 opérateurs :

1. 3 métriques influençant sur notre choix du voisin : le coût d’une arête (la distance entre i et j fois le nombre de fois qu’une arête a été pénalisé = considérée comme la pire), la profondeur d’une arête (distance entre le point le plus éloigné du dépôt et le dépôt), la largeur d’une arête (différence de longueur entre les projetés de i et j sur la droite passant par le dépôt et le centre de gravité de la tournée. Ce CDG s’obtenant en faisant la moyenne des projetés des points de cette tournée. La formule est :

b(i,j) =

1. Ejection-Chain : Pour commencer, on cherche le plus proche voisin j dans la tournée 2 de i dans la tournée 1. On doit pouvoir mettre i dans la tournée 2. Ensuite, on insère i après j dans la tournée 2. Ainsi cela fait (dans la tournée 2), point j, puis point i, puis point j etc…. Cela nous permet donc de ne pas faire de détour car la distance est moins longue lorsque le point i va dans la tournée 2 plutôt que dans la tournée 1 où il y a un détour. Le point i n’étant plus dans la tournée 1, on réalise cela jusqu’à atteindre la tournée 1 (on enlève un point i dans la tournée 2 pour le mettre dans la tournée 3).
2. Cross-Exchange : Cet opérateur permet d’empêcher les arêtes d’une tournée de croiser avec les arêtes d’une autre tournée car d’après le problème du plus court chemin, on peut constater que si le chemin le plus court se ne croise jamais.
3. Lin-Kernighan : Tout d’abord, cet opérateur se focalise tournée par tournée. Chaque tournée possède k clients et peut réaliser k-opt, c’est-à-dire le fait d’échanger k clients différents sur la tournée. Première tournée, on applique un 2-opt donc on échange les 2 clients entre eux. S’il y a une amélioration, on passe à 3-opt et ainsi de suite. Par contre, dès le moment où il n’y a pas d’amélioration, on retourne à 2-opt et on change de combinaison.

* Algorithme d’optimisation utilisé : La condition d’arrêt de la boucle n’est plus 3 minutes mais peut varier en fonction des instances. Cette fois-ci le CE et le EC sont remplacés par le mode FI-RD qui permet de faire un voisinage de solutions de manière aléatoire et prendre le premier voisin qui améliore la solution courante. Ensuite, dans cet algorithme, on rajoute une condition *Restart* qui s’active lorsqu’il n’y a pas eu d’amélioration depuis *restartTime* secondes. Cette condition permet de conserver un % de meilleure solution parmi toutes les arêtes et de relier les arêtes entre elles en fonction des clients qu’elles ont en commun pour en faire des tournées. Et pour finir on applique l’algorithme CW sur les tournées obtenues.

1. **Lecture algorithme pour problème de tournée de véhicules :**

Seulement, on peut relever une grosse problématique : tous les algorithmes que l’on a vu jusqu’à présent ne prenne en compte qu’un seul dépôt or nous avons besoin de pouvoir prendre en compte une multitude de dépôt. C’est pourquoi nous avons pensé qu’il serait judicieux de faire dépôt par dépôt et non faire un algorithme avec plusieurs dépôts. Pour cela, on va faire du découpage. On peut par exemple, prendre pour chaque dépôt, un découpage de 20km autour du dépôt. Et si un client est recouvert par deux découpages de dépôts, alors on affecte le client au dépôt le moins loin.

Cependant, pour faire cela, déjà il faudrait pouvoir calculer la distance entre chaque dépôt et chaque client et il faudrait aussi pouvoir savoir quels dépôts nous sélectionnons.

**Idée :**

Pour sélectionner les entrepôts, on pourrait faire de cette façon : on prend chaque entrepôt et on fait un découpage de α km. S’il reste beaucoup trop de magasin qui ne sont pas compris dans le découpage d’un entrepôt on augmente la taille du découpage. S’il n’y a que quelques magasins qui ne sont pas compris dans un découpage, on peut faire en sorte qu’un entrepôt ait un découpage partiel plus grand pour ce magasin en fonction du dépôt le plus proche. Après avoir fait le découpage de tous les entrepôts, on enlève tous les entrepôts qui possèdent une quantité β (ça peut être 3 magasins en commun avec les autres par exemple) de magasins en commun avec le découpage d’autres entrepôts (en commun = dans le découpage d’un autre entrepôt). Cela permettra ainsi de diminuer la quantité d’entrepôt en enlevant les entrepôts inutiles.

Cette idée de suppression des entrepôts avec une quantité β trop élevé est une mauvaise idée car il n’existe déjà que trop peu d’entrepôt (24). Si l’on venait à en supprimer, l’écart entre certains magasins et le dépôt le plus proche serait trop élevé. Cependant, cette quantité d’entrepôt pourrait couter chere, on pourrait alors augmenter β afin qu’il y est peu d’entrepôt qui soit enlevé, seulement les vrais inutiles, ou alors on pourrait diminuer β afin d’enlever un maximum d’entrepôt pour que ça ne coute pas chère. C’est donc une question complexe qui demanderai un algorithme avec comme objectif de minimiser la quantité d’entrepôt.

Il faudrait ainsi **1 objectif : Minimiser le coût, donc la totalité des frais doit être minimiser tout en ayant comme contrainte le fait que la totalité des frais doit être inférieur à une somme maximum.**

**NOS INFORMATIONS -----------🡪**

* Les frais : 70€/jour + 0.40€/km + 10€/panier repas + frais parking péage remboursé sous présentation ticket
* Il est possible qu’il n’y ait pas d’entrepôt donc la zone (le découpage) sera par magasin et on doit trouver un livreur dans la zone
* Il y aura des entrepôts à priorisé, à faire en premier
* Un véhicule est directement associé à un livreur
* 1 livreur peut livrer 6 magasins alors qu’un autre livreur peut en livrer 1
* Il y a deux types d’entrepôts
* Pas besoin de sélectionner des entrepôts car on les a tous dans tous les cas
* L’objectif est :

C’est-à-dire minimiser le coût, même s’il restera dans tous les cas inférieurs au maximum du budget possible

**NOS CONTRAINTES ----------🡪**

* Il n’y a pas de limite de stock dans l’entrepôt ou dans le véhicule du livreur
* Il peut y avoir un bénéfice à devoir faire dans un délai de temps (ex : 45 000€ en 22 jours)
* La somme des frais est :

**Donc**

**Autre idée concernant l’attribution des magasins à un entrepôt :**

Initialement, nous avions l’idée de faire un périmètre à vol d’oiseau autour de chaque entrepôt. LE problème auquel nous pensons est le suivant : suivant le type de route qui sépare le point de livraison et l’entrepôt (autoroute, centre-ville, nationale,…) , le temps de trajet ne sera pas du tout le même. A vol d’oiseau, on peut faire 10km d’autoroute en 10min, alors que traverser une ville de 10km prendra 45min.

L’idée pour pallier à ce problème, est de faire un découpage autour de l’entrepôt en fonction du temps de trajet.

Exemple : je prends l’entrepôt A, et je vais lui attribuer tous les magasins qui sont à moins de deux heures de routes de cet entrepôt. Si un magasin se trouve à plus de deux heures de tous les entrepôts, on va l’attribuer à l’entrepôt le moins loin.

Pour faire ces découpages, nous avons besoin des temps de trajets calculés entre chaque entrepôt et chaque magasin. (on aura donc n entrepôts x m magasins =nm trajets possibles).

On décide de fixer le seuil à deux heures de trajets depuis l’entrepôt. Si un point se trouve à moins de deux heures de l’entrepôt, on va le rattacher dans la zone de cet entrepôt. Le point jaune se trouve à plus de deux heures de tous les autres entrepôts. On va quand même m’intégrer à la zone rouge car ce magasin est plus proche de l’entrepôt de la zone rouge que de l’entrepôt de la zone en jaune.

Nous avons pensé aussi que si un magasin est à plus de 2 heures de route de tout entrepôt et donc qu’il est tout seul, qu’on pourrait le relier au magasin le plus proche plutôt qu’à l’entrepôt le plus proche. Cela permettrait d’éviter, un aller-retour juste pour un seul magasin ainsi que d’utiliser un livreur pour un seul magasin. Même si cela va devoir coûter en complexité car si on suit cette solution, le seul moyen de connaître l’itinéraire entre chaque magasin est de faire un graphe complet ce qui fait trajets possibles. En comptant en plus de cela les itinéraires entre les entrepôts et les magasins. Pour se rassurer, on peut se dire que dans tous les cas, chaque graphe complet sera découpé en plusieurs graphe complet en fonction de la zone de l’entrepôt amenant à une recherche par saccades. Pour l’opération Chanel, il y a 919 magasins. Si on divise par le nombre d’entrepôt qui est 24, on obtient environ 38. Dans chaque zone (environ), on aura 38 magasins ce qui amènera (avec un graphe complet) à 1444 trajets possibles. + les trajets entrepôts->magasins égales à 912 trajets en plus. Cela nous fait environ 3800 trajets possibles par zone, à nous de décider si cela revient à un temps de complexité trop long ou pas.

**Idée :**

Après des recherches, on a compris que pour notre problème, il fallait faire un algorithme glouton, c’est-à-dire un algorithme qui étape par étape la solution qui lui semble la plus optimale. Donc on optimise chaque étape en la minimisant en coût sans jamais remettre en cause le choix précédent.