Université de Nantes — UFR Sciences et Techniques
Master informatique parcours “optimisation en recherche opérationnelle (ORO)”
Année académique 2019-2020

Optimisation discrète et combinatoire

Dossier d’étude

Etude de la résolution du TSP-D

Nilson Toula

8 janvier 2020

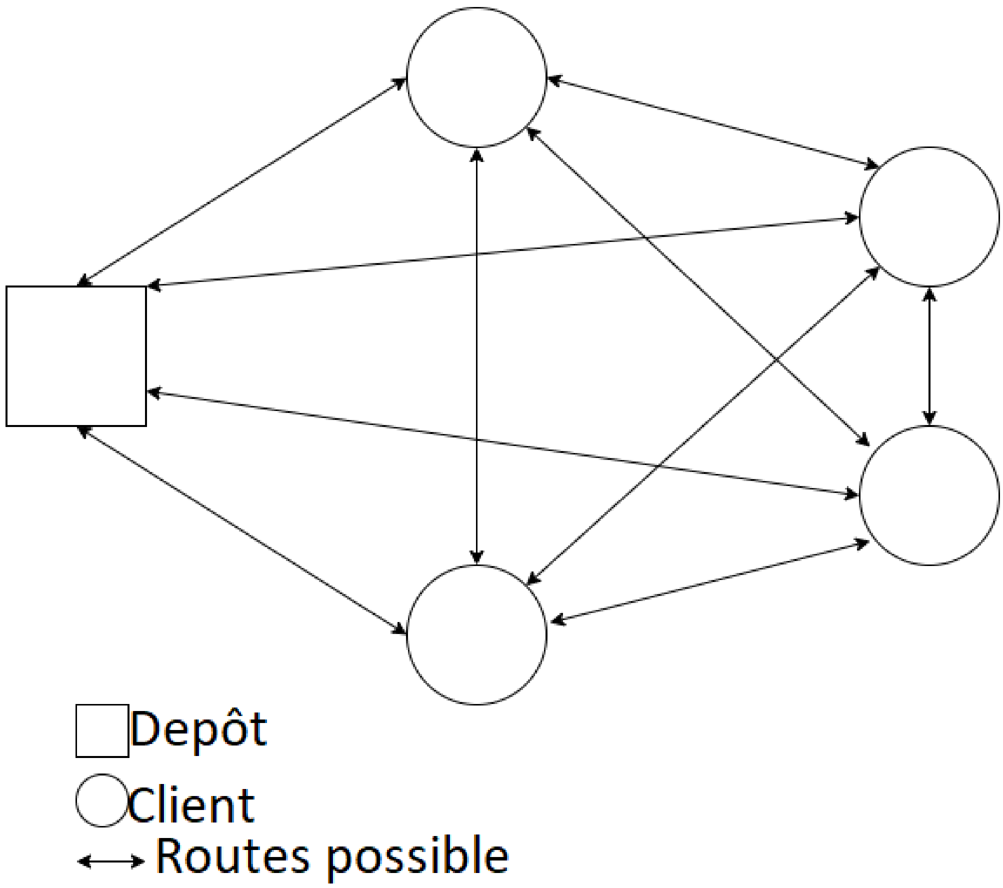
Résumé du dossier d'étude

Nous étudierons ici le problème du TSP-D, plus particulièrement, nous proposons une démarche pour réaliser sa résolution exacte via la méthode dite AEP.

1 Définition du problème

— Le problème étudié est une variation du **problème de voyageur de commerce** [1].

Première approche :
Un postier doit déposer des colis certains habitants d'une certaine zone, pour se faire, il part du bureau de poste, livre les colis à ses clients et retourne au bureau de poste après sa tournée.
Pour l'aider dans sa tache, il dispose d'un drone autonome lui permettant de livrer un colis à la fois.
Quel est le meilleur itinéraire pour le postier et à quel moment doit-il utiliser son drone ?



Dans cette étude nous allons principalement nous basé sur deux articles, que nous appelleront familièrement par le noms de leur premier auteur, celui de Agatz [2] et celui de Poikonen [3]
Leur article répond à la demande croissante des services d'achat sur Internet, recherchant de "nouvelles technologies pour effectuer les derniers kilomètres".

1. aussi appelé Traveling Salesman Problem (TSP)
2. Référence de l'article en bibliographie
3. Référence de l'article en bibliographie

1.1 Paramètres

Pour résoudre ce problème, nous dépendrons des paramètres suivants :

— D'un distancer sous la forme d'une matrice, représentant les distances (Euclidiennes) entre les **n-1** points de livraison.

— Un ordre de visite des différents clients, obtenue par résolution exacte du problème sans inclure l'utilisation du dorne
(ie : la solution excate du problème de TSP associé) .

— D'un camion de vitesse V_{camion} capable d'effectuer tous les chemins possible sur entre nos clients.

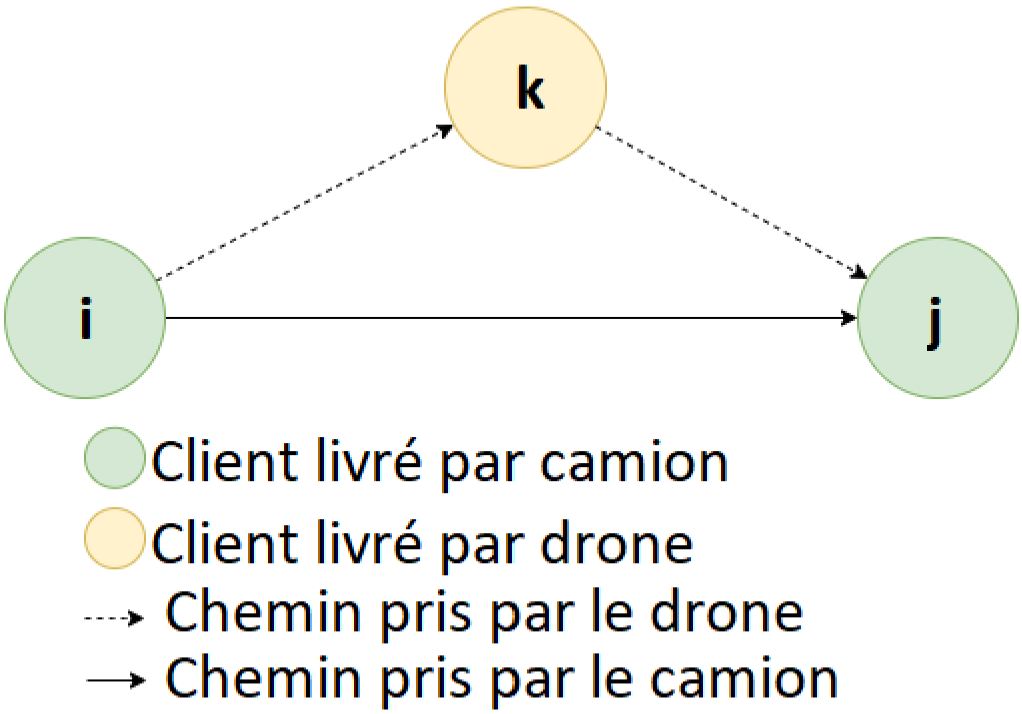
— D'un drone de vitesse V_{drone} capable d'effectuer toutes les livraison que doit faire le camion.

— D'un set d'instance type contenant chacune les coordonnées des différents points à livrer, ainsi que les vitesses respectives du drone et du camion.

Nous retournerons l'ordre de visite du drone et du camion sous forme de suite d'opération (voir la section suivante) ainsi que le temps total du parcours.

1.2 Opération (i,j,k)

Une opération est l'action de séparation du drone et du camion.
— L'indice i représente les point de départ du drone.
— L'indice j représente le point de rendez-vous entre le drone et le camion.
— L'indice k représente le point de livraison du drone.
Nous noterons que pour toute opération : $i < k < j$.
De plus, nous indiquerons que k=0 quand le camion effectue seul toutes les livraisons des clients i à j.



.

2 Démarche de résolution

2.1 Analyse du problème et limitation

Pour cette résolution nous nous contentons d'améliorer une solution de TSP déjà existante, ce parti pris nous prive de la réelle solution optimale de l'instance pour le problème de TSP-D.
En effet, rien ne nous garantie que l'ordre de visite du TSP-D serait le même que celui du TSP.
De plus nous admettons que le camion ne reste pas stationnaire. Ce choix, comme préciser dans l'article de Poikonen, peut nous empêcher d'avoir une la solution optimal de l'instance du TSP-D.

2.2 Création d'un distancier

Nous disposons d'instance standards accompagnant les articles d'Agatz et de Poikonen.
Les instances d'Agatz sont sous la forme d'un document texte comportant les vitesses du drone et du camion, ainsi que les coordonnées des différents points de livraison.
Pour traité notre problème nous établiront alors un distancier indiquant la distance relative d'un client par rapport à tous les autres.
Pour ce faire nous utiliseront une métrique Euclidienne [4], qui nous permet de tester notre démarche sans perdre en généralité.

2.2.1 Coordonnées et distance Euclidienne

Soit a et b deux point d'un plan P, respectivement de coordonnée (x_a,y_a) et (x_b,y_b).
La distance Euclidienne entre ces deux point [5] est définie comme suit :

$$\boxed{\forall(a,b) \in P : d(a,b) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}}$$

Nous créons donc le distancier comme suit :

Algorithm 1 calculDistancier

```
Input :
    list : pos // Liste des coordonnées sous forme de couple (x,y)
Var :
    matrix : distancier // Matrice n * n, avec n le nombre de nodes à visiter
for all i in pos do
  for all j in pos do
    distancier[i,j] ← d(i, j)
  end for
end for
return(distancier)
```

4. Familièrement appelé "Distance à vol d'oiseau"
5. Aussi appelé "2-distance"

2.3 Résolution exacte du problème de TSP

Pour réaliser une solution exacte du TSP, nous utiliseront le solver GLPK [6].
Pour se faire, nous devons formaliser notre problème sous la forme d'un programme linéaire à l'aide du langage de modélisation JuMP [7].

2.3.1 nécessite du passage par une résolution de LAP

Le problème du voyageur de commerce se décompose suivant différentes programmation linéaire. Cependant celle-ci demande un nombre de contrainte en $O(n^2)$ avec n le nombre de nodes/clients à visiter.
Une des modélisation reconnue de ce problème est celle de Miller-Tucker-Zemlin, qui à pour but d'illustrer le nombre de contraintes associées :

$$
\begin{aligned}
\min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} & c_{ij} x_{ij} : & \\
& x_{ij} \in \{0,1\} & i, j = 1, \ldots, n; \\
& u_i \in \mathbf{Z} & i = 2, \ldots, n; \\
& \sum_{i=1, i\neq j}^{n} x_{ij} = 1 & j = 1, \ldots, n; \\
& \sum_{j=1, j\neq i}^{n} x_{ij} = 1 & i = 1, \ldots, n; \\
& u_i - u_j + n x_{ij} \leq n - 1 & 2 \leq i \neq j \leq n; \\
& 0 \leq u_i \leq n - 1 & 2 \leq i \leq n.
\end{aligned}
$$

[8]

Avec $x_i j$ le choix d'emprunter l'arc reliant la node i à la node j.

Nous avons donc décider de passer par une résolution d'un problème "plus simple" et moins contraint : le LAP [9], auquel nous répondrons pour ensuite modéliser un problème de TSP par ajout successif de contraintes.

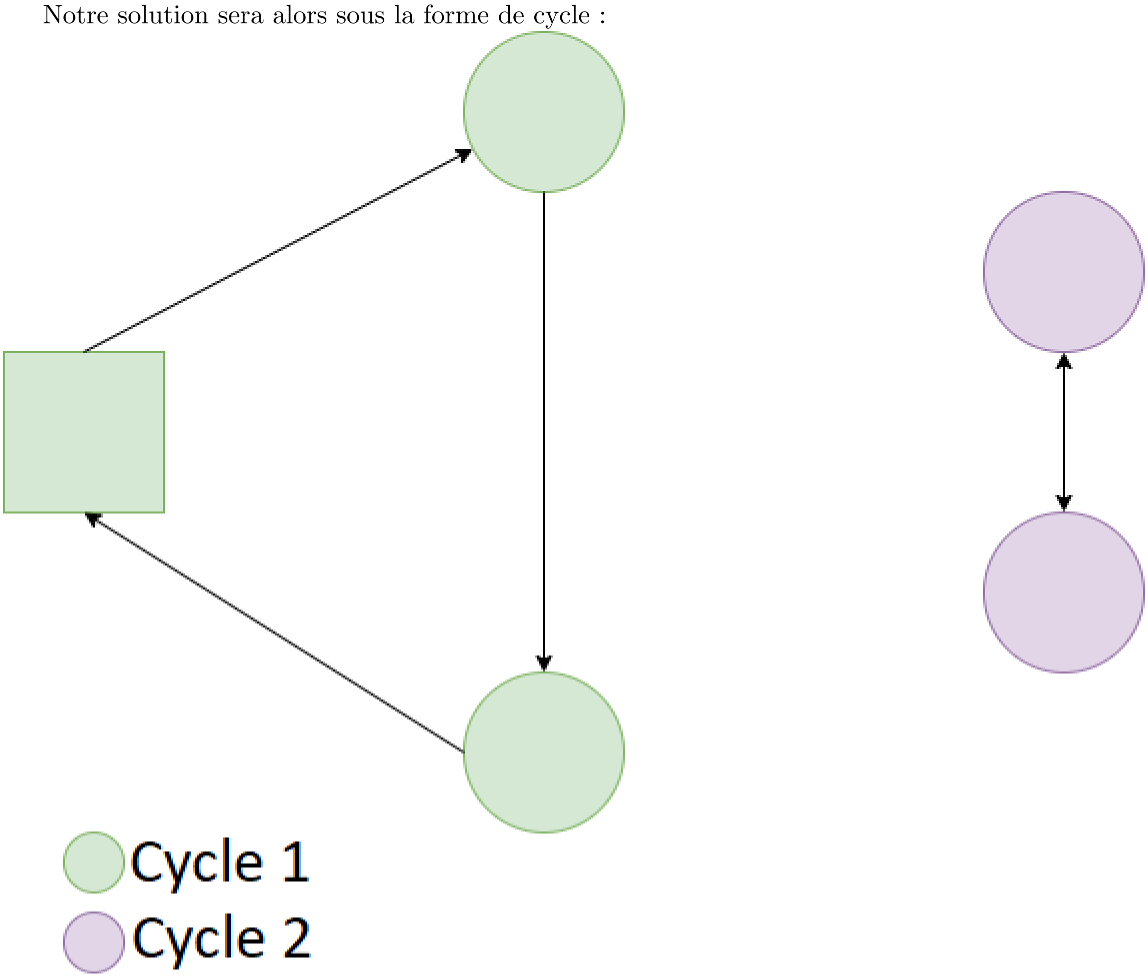
6. https ://www.gnu.org/software/glpk/
7. https ://github.com/JuliaOpt/JuMP.jl
8. Figure issue de l'article Wikipedia sur le TSP
9. linear assignment problem

2.3.2 Modelisation du LAP

Nous utiliseront pour se faire la modélisation proposé par X.Gandibleux dans son cours d'optimisation discrète et combinatoire :

$$\left[\begin{array}{llll} \min z & = & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\ s/c & & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j = 1, \ldots, n \\ & & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1, \ldots, n \\ & & x_{ij} = (0,1) & \begin{array}{l} i = 1, \ldots, n \\ j = 1, \ldots, n \end{array} \end{array}\right]$$ [10]

Notre solution sera alors sous la forme de cycle :



10. Issue du chapitre 4 du cours d'optimisation de X.Gandibleux de 2019

2.3.3 supression des cycles

La modélisation vu ci-dessus nous retourne une solution sous forme d'une matrice que nous exploitons pour trouver des cycle d'une permutation.
Pour se faire, pour chaque cycles trouvé, nous ajoutons une contrainte de sorte que :
$\forall x_{ij} \in P : \sum x_{ij} < |P| - 1$, avec P le cycle à supprimer et $|P|$ son nombre d'éléments.

Nous utiliseront donc l'algorithme suivant :

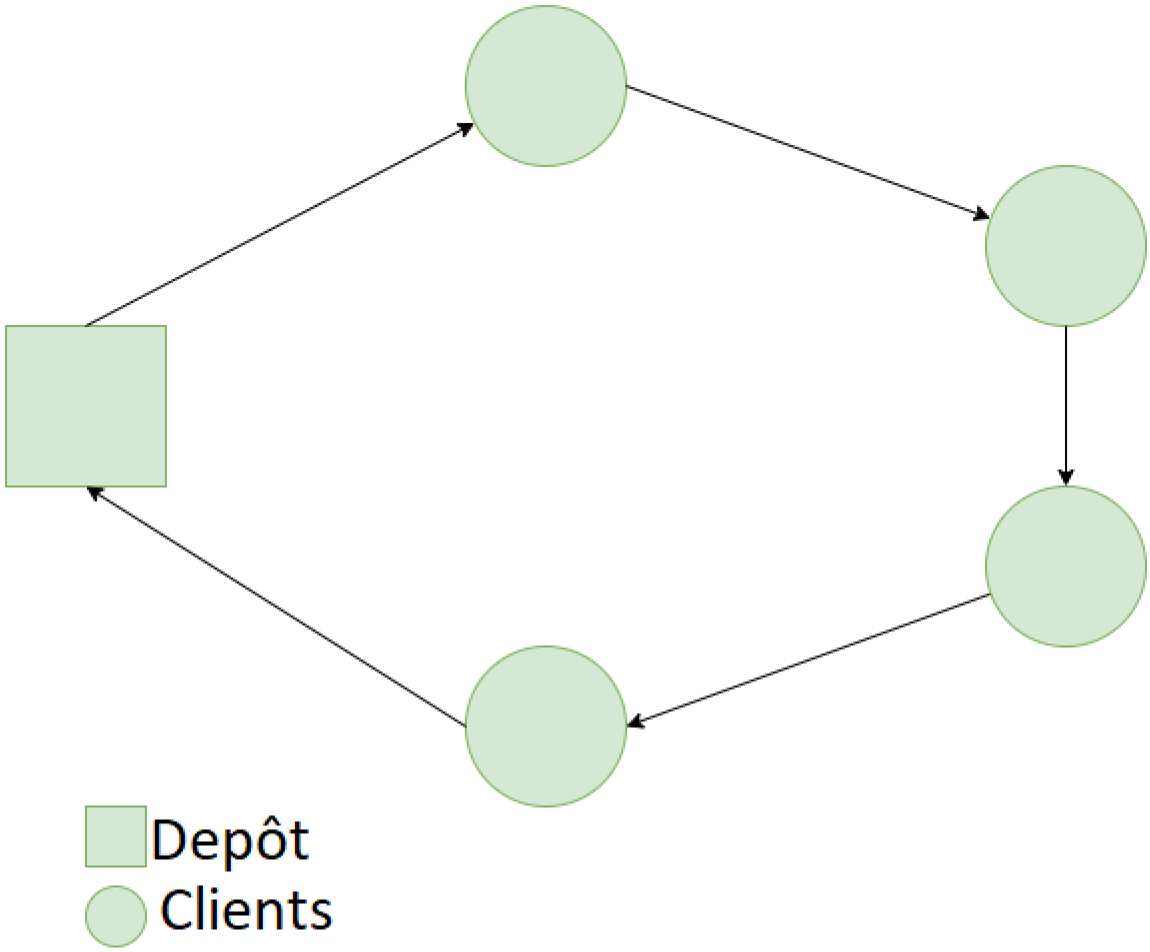
Algorithm 2 setTSP

Input :
  linear model : ip // Notre modèle linéaire
  matrix : x // La solution associé à ce problème

souscycle ← $getSousCycle(ip, x)$
while $|souscycle| > 1$ **do**
  for p **in** souscycle **do**
    for all x **in** p **do**
      ip ← *add*(*ip*,$\sum x_{ij} < |P| - 1$)
    end for
  end for
  solve(ip,x)
  souscycle ← $getSousCycle(ip, x)$
end while
return(ip,x)

2.4 Mise en place d'une résolution exacte du problème de TSP-D

Nous décrirons ici la démarche de résolution du TSP-D avec la méthode AEP [11] qui consiste en un partitionnement exacte de le solution, basé sur une programation dynamique.
La solution initiale sera de la forme :



2.4.1 Calcul de toutes les opérations

Nous procédons d'abord à un calcul de toutes les opérations (i,j,k) possible, en stockant celles-ci dans une liste de matrice [12] que nous appellerons *tempsOp*.
L'indice k de l'opération sera le même que celui de la liste, et les i, j correspondant seront les coordonnées du temps de l'opération.

2.4.2 Calcul du meilleur temps entre T(i,j) pour tout i,j

Nous cherchons alors dans la matrice *tempsOp* le meilleur temps quelque soit le k pour toute opération (i, j, k). Nous mémoriserons ces informations dans deux matrice M et P, conservant respectivement temps de cette opération, et le k correspondant.

11. AEP pour Agatz Exact Partioning décrite dans l'article de Agatz et nommé ainsi dans celui de Poikonen
12. Ces matrices seront toutes des matrices triangulaires supérieur

2.4.3 Programmation dynamique

Tous comme décris dans l'article de Agatz, nous procédons au calcul de la meilleur suite d'opération.
Pour se faire, nous rempliront deux vecteur V et K, comme suit :

Algorithm 3 meilleurSuiteOperation

```
  Input :
        list : odrePassage // Solution de notre TSP
        matrix : M // Valeur minimal de l'opération (i,j) pour tout k
        matrix : valK // k associé à l'opération minimal de M
  Var :
        list : V
        list : P
        float : valMin
        int : kMin
  V[1] ← 0
  P[1] ← 0
  valMin ← ∞

for i from 2 to length(ordrePassage)-1 do
    for k in 1 to i-1 do
      if V[k] + tempsMin[k,i] ¡ valMin then
          valMin ← V[k] + tempsMin[k, i]
          kMin ← k
      end if
    end for
    V[i] ← valMin
    P[i] ← kMin
    valMin ← ∞
end for
return(V,P)
```

3 Algorithmes

Voici l'algorithme général nous permettant d'effectuer le traitement du TSPD :

Algorithm 4 main

```
  Input :
        text document : instance
  ip, x ← parse&modliseLAP(instance)
  ip, x ← setTSP(ip, x)
  tempsOp ← calculToutesOperation(ip, x)
  tempsMin, valK ← calculMeilleurTemps(tempsOp)
  V, P ← meilleurSuiteOperation(tempsMin, valK)
  synthèse(V,P,valK)
return(V,P)
```

La synthèse nous permet de déterminer les opérations effectués.
Nous procédons comme suit :

Algorithm 5 synthèse

Input :

list : V // liste des meilleurs temps pour arriver à cet élément de la liste

list : P // chaque élément sera la node de départ de l'opération ayant mené à cet élément de la liste

matrix : valK // k associé à l'opération minimal de M

Var :

int : i

int : j

int : k

int : cpt

i ← $P[End]$

j ← $length(V)$

K ← $valK[i, j]$

print("Première opération : " (i, j, k))

cpt ← 1

while i =! 1 **do**

j ← i

i ← $P[i]$

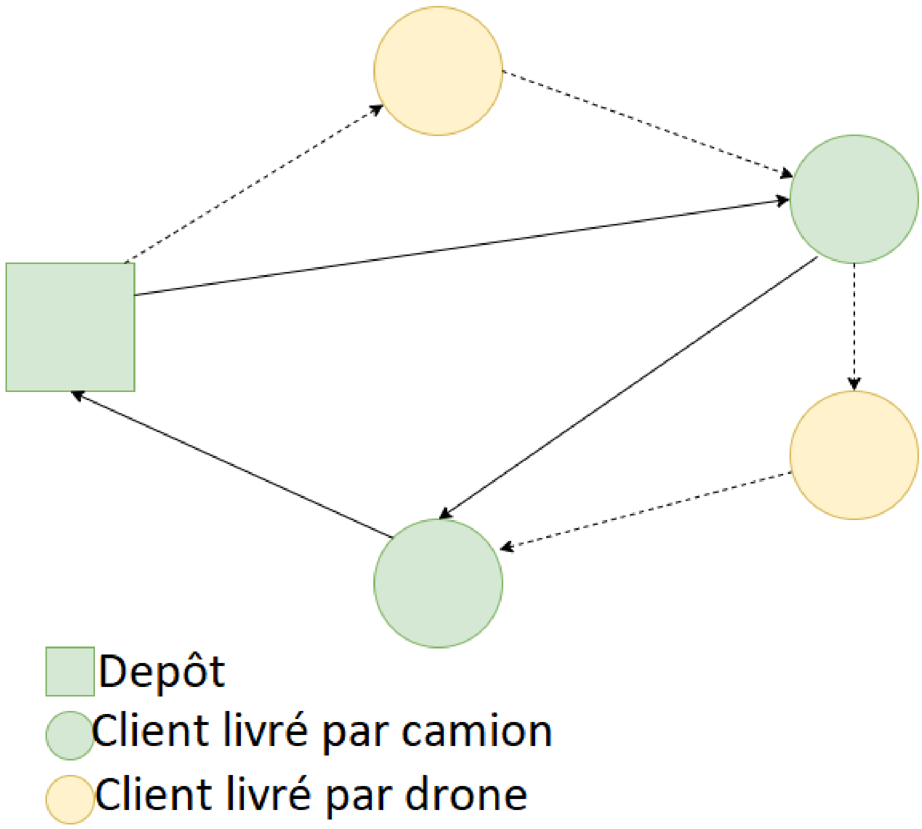
k ← $valK[i, j]$

print("Opération numéro : ", cpt, " ", (i,j,k))

cpt++

end while

Après traitement, la solution sera de la forme :



4 Contexte expérimental

La machine utilisé pour cette expérimentation possède les paramètres suivants :

Édition Windows

Windows 10 Professionnel

© 2018 Microsoft Corporation. Tous droits réservés.

Système

Processeur :	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
Mémoire installée (RAM) :	8,00 Go (7,85 Go utilisable)
Type du système :	Système d'exploitation 64 bits, processeur x64

Nous nous utiliserons les instances suivantes :
- doublecenter-53-n10.txt
- doublecenter-56-n10.txt
- singlecenter-53-n10.txt
- singlecenter-56-n10.txt
- uniform-53-n10.txt
- uniform-56-n10.txt

5 Expérimentation

Résultat de notre expérimentation (le temps est une moyenne de 10 itérations par instances) :

Instance	Nombre d'itérations pour suprimer les cycles des LAP	Solution TSP	Solution aep-TSPD	Temps d'exécution (en secondes)
doublecenter-53-n10.txt	3	797.96	571.91	0.049373
doublecenter-56-n10.txt	20	671.44	523.25	0.516167
singlecenter-53-n10.txt	3	510.45	320.43	0.040053
singlecenter-56-n10.txt	3	532.15	245.48	0.041078
uniform-53-n10.txt	2	284.65	236.85	0.030948
uniform-56-n10.txt	3	322.64	252.14	0.038764

Etant donné la taille des instances (ayant chacune 10 nodes) le temps d'exécution est très peut élevé. Cependant on observe que le nombre de cycle à supprimer pour transformer le LAP en TSP joue beaucoup sur celui-ci, avec un temps presque vingts fois supérieur entre l'instance "doublecenter-56-n10.txt" et "uniform-53-n10.txt".

6 Conclusion générale

La méthode de résolution AEP est une méthode permetant une amélioration significative du temps de parcours comparer vis à vis de celui d'un TSP classique.
Notre expérimentation se base sur des données générales pouvant être affiné au besoin affin de mieux correspondre à la réalité.
Nous pourrions par exemple :

— Envisager une métrique différente entre le drone et le camion (celui-ci devant emprunter les routes et donc ne suivant pas une métrique Euclidienne).

— Ajouter une autonomie au drone, ainsi qu'une charge maximale.

— Ajuster la vitesse du camion entre deux nodes (représentant la différence entre les différentes limitations de vitesse et état du trafic).

Une autre limitation est l'obligation que le Camion ai à se déplacer entre deux opération, lever celle-ci pourrait encore plus améliorer le temps de parcours (c'est ce que propose Poikonen dans son article).
De plus, l'ordre de parcours imposer par la solution exacte du TSP ne garantie pas que celle-ci soit optimale même avec le fait que le camion puisse rester stationnaire.

Bibliographie

— **Article de Agatz :**
Optimization Approaches for the Traveling Salesman Problem with Drone Niels Agatz, Paul Bouman, and Marie Schmidt *Transportation Science 2018 52 :4, 965-981.*
https ://doi.org/10.1287/trsc.2017.0791

— **Article de Poikonen :**
Stefan Poikonen, Bruce Golden, Edward A. Wasil (2019) **A Branch-and-Bound Approach to the Traveling Salesman Problem with a Drone.** INFORMS *Journal on Computing 31(2) :335-346.*
https ://doi.org/10.1287/ijoc.2018.0826

— **Le Linear Assignement Problem (LAP) :** https ://en.wikipedia.org/wiki/Assignment$_{p}roblem$

— **Le Traveling Salesman Problem (TSP) :** https ://en.wikipedia.org/wiki/Travelling$_{s}alesman_{p}roblem$

— **Cours de Master 1 en Optimisation Discrète et Combinatoire du Pr. X.Gandibleux et A.Przybylski** pour l'année universitaire 2019-2020 à l'université de Nantes