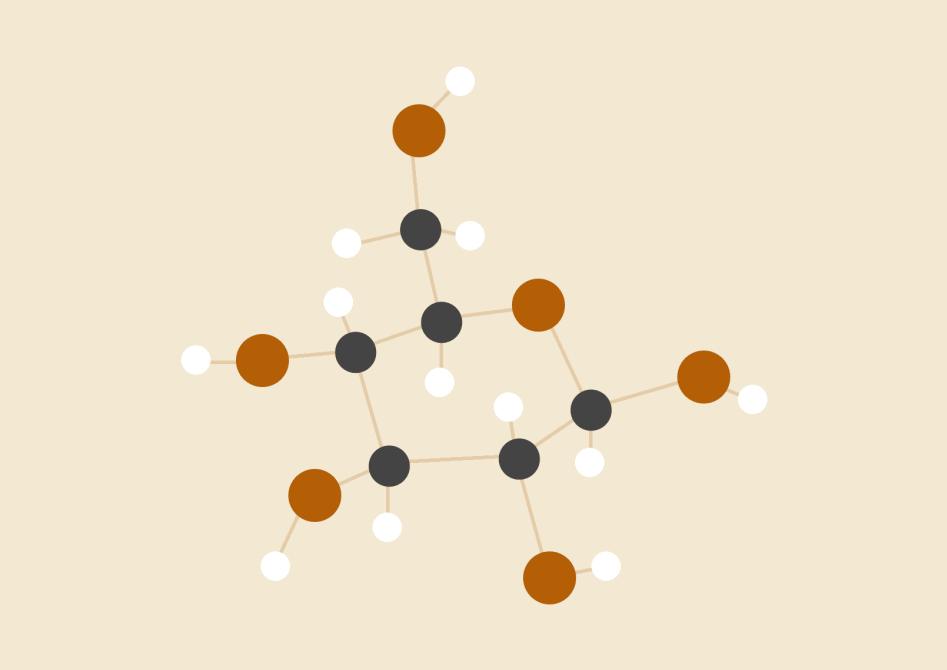
Rapport Intelligence Collaborative

Problèmes des tournées de véhicules avec fenêtres de temps (PTVFT)



groupe ICOmprendRien

Andreis Purim
Mehdi Hammas
Colin Prudhomme
Yaniv Benichou
Yohann Le Couster
James Maistret

29/04/2022

Sommaire

Introduction

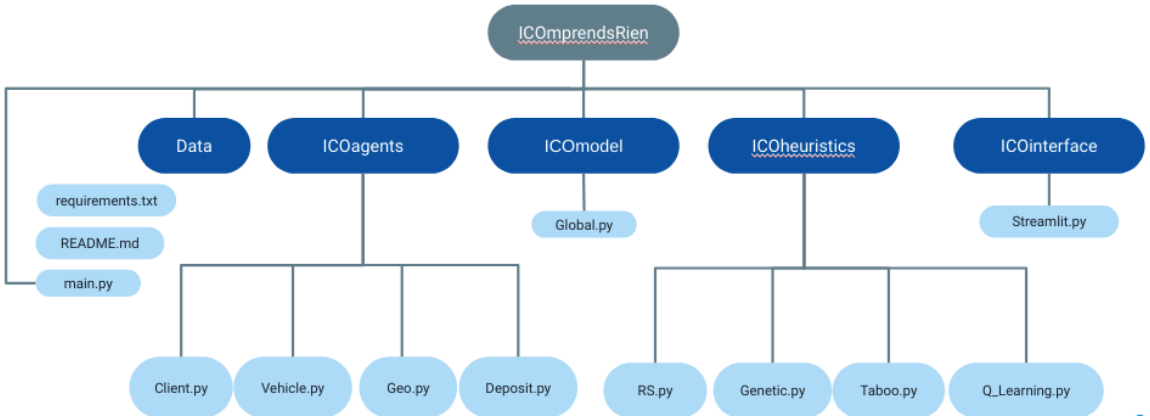
Notre projet pour l'électif d'Intelligence Collaborative, a pour thème, le problème des tournées de véhicules, qui est une généralisation du problème du voyageur. Notre but est d'optimiser le parcours des véhicules entre le dépôt et les clients, en prenant en compte plusieurs éléments différents, et en explorant différentes méthodes que nous explorerons le long de ce rapport. Classé dans la catégorie de complexité "np-difficile", le problème des tournées de véhicules représente un enjeu majeur pour l'optimisation des trajets quotidiens que font tous les travailleurs sur la route.

Nous commencerons notre rapport en présentant la manière dont nous avons géré notre projet et notre équipe, puis une description plus détaillée du problème avant d'introduire la Data Vizualisation qui nous a été très utile en amont du projet. Ensuite, nous détaillerons un à un, les différents algorithmes d'optimisation par métaheuristiques : Tabou, Recuit Simulé et Génétique, et par collaboration à l'aide des systèmes multi-agents. Enfin, l'apprentissage par renforcement ainsi que la comparaison de tous les résultats obtenus nous permettront de conclure ce rapport.

I. Gestion de projet

Pour notre projet, nous avons décidé d'utiliser Github, une plateforme permettant de maintenir à toute étape du projet, une version stable du code, avec un historique des changements effectués et une communication sur ces derniers. Au total, ce seront près de 140 commits qui auront été effectués, ainsi que 3 branches, montrant l'investissement et la bonne utilisation que les membres de notre groupe ont pu en faire.

Nous avons mis en place également un fichier requirements.txt contenant l'ensemble des librairies nécessaires à installer pour faire correctement fonctionner notre code. De plus, une documentation d'exécution est disponible, dans le fichier README.md . Enfin, l'arborescence des fichiers de notre code utilise des mots clés, permettant plus de clarté pour mieux s'y retrouver. Voici à quoi ressemble notre arborescence finale :



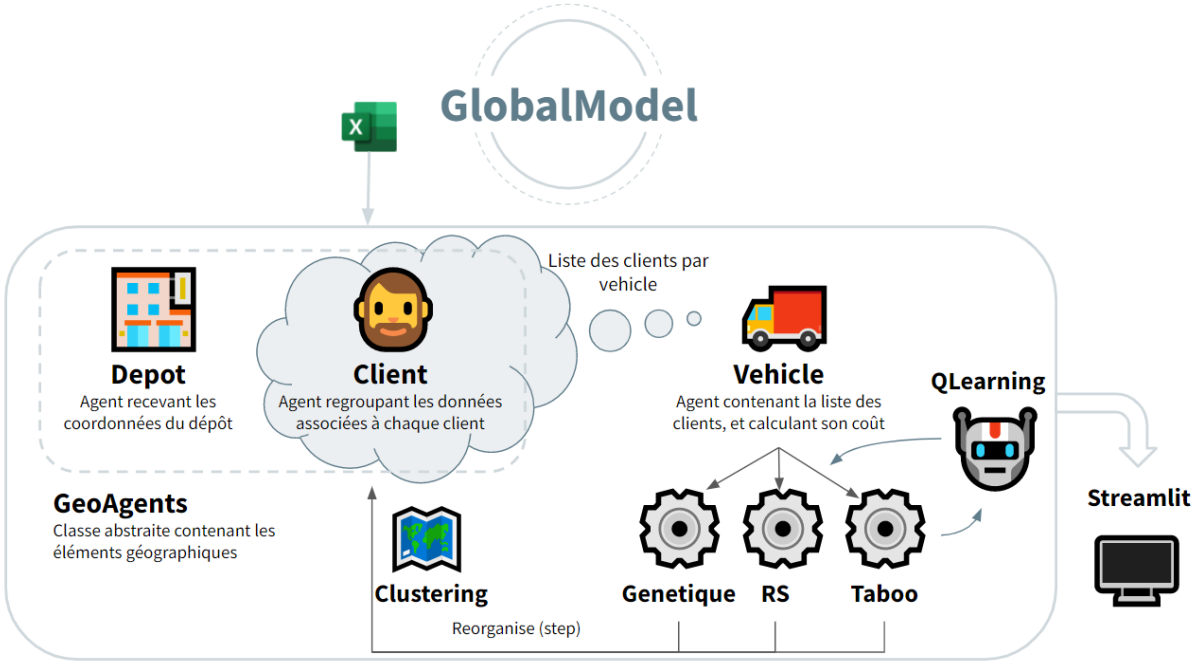
arborescence des fichiers

Chacune des cinq branches principales segmente le fil rouge de façon à ce que chacune d'elles se module bien. Les fichiers .py peuvent être détaillés de la sorte :

- main.py est le cœur du fil rouge, là où se lance l'ensemble du projet.
- Client.py décrit l'ensemble des caractéristiques pour décrire chaque client
- Vehicle.py, de même, rassemble les instructions pour chaque camion
- Geo.py permet de déduire la distance entre deux points décrits par leurs coordonnées géographique (longitude, latitude)
- Deposit.py rassemble l'agent pour le dépôt.
- Global.py permet de lire les bases de données sous format csv, d'attribuer les clients

D'autre part, nous avons mis en place des documents partagés de répartition des tâches, permettant à chacun de comprendre le travail restant à faire et à répondre aux principales questions : " qui ? quoi ? quand ? " .Cette répartition s'est avérée être très efficace, car , accompagné de réunions régulières, personne ne s'est retrouvé ni perdu ni délaissé quant à l'avancée du projet.

II. Méthodologie de détermination des solutions



Notre approche à la résolution du problème se base sur le constat qu'une solution est constituée d'un ensemble de véhicules, auxquels sont attribués une liste de clients dont l'ordre correspond à l'ordre de parcours du véhicule. Notre recherche de solutions se base donc sur deux processus : la distribution des clients aux divers véhicules puis l'optimisation du parcours de chaque véhicule parmi ses clients.

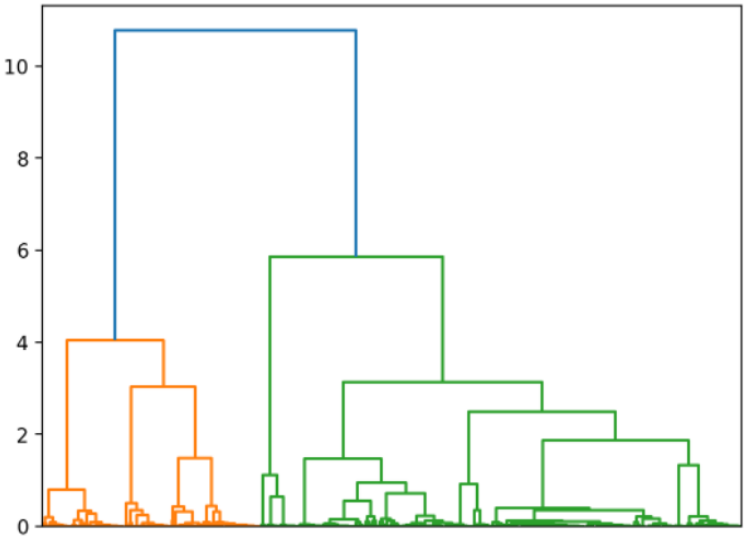
A. Clustering et première attribution des clients

Étant donné que notre solution dépend de la réduction de la distance que chaque véhicule doit parcourir, il est naturel de supposer que les véhicules prendraient de préférence les clients qui se trouvent les uns à côté des autres. Cette idée est renforcée lorsque nous visualisons les données sur une carte et réalisons que de nombreuses livraisons sont regroupées dans les villes et les villages, l'espace entre les deux n'étant pas utilisé.

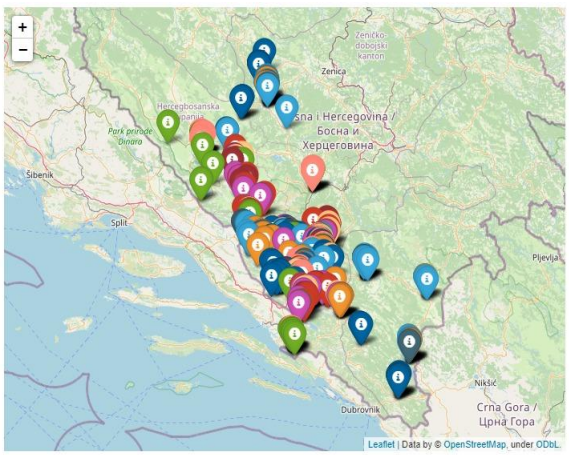
Notre idée était donc d'adopter une approche naïve en regroupant géographiquement les clients. Cela ne ferait pas nécessairement les meilleurs itinéraires clients mais indiquerait aux algorithmes d'optimisation quels sont les clients les uns à côté des autres. Cela permet déjà de commencer avec une bonne solution et de continuer à optimiser l'espace de solution.

Le clustering utilisé était l'AgglomerativeClustering de la librairie SkLearn. L'idée

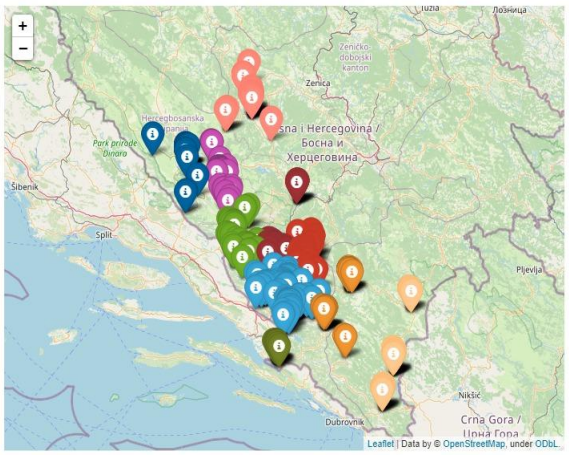
est que l'algorithme de clustering commencera de manière récursive à diviser les clients en groupes jusqu'à ce qu'ils deviennent unitaire. Après cela, l'algorithme commence à regrouper les clients les uns à côté des autres en tenant compte du poids et du volume. Il les regroupera jusqu'à ce que le plus grand groupe ne puisse plus rentrer dans un véhicule. Cet effet est visible sur le graphique suivant (dénogramme des clusters)



Et maintenant, nous pouvons comparer l'apparence de nos clients tels qu'ils ont été extraits de la base de données et comment ils sont regroupés par couleur après le regroupement:



Avant



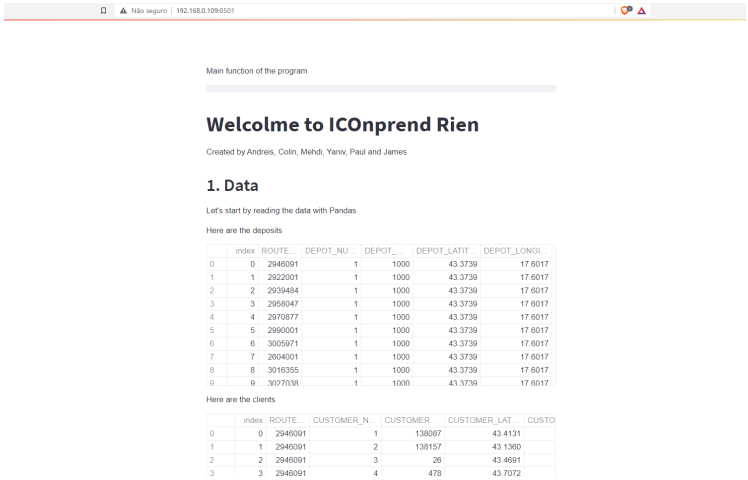
Après

Sur ces images, on voit bien qu'avant le clustering, les marqueurs étaient

désordonnés. Tandis qu'après, les clusters sont évidents. Chaque cluster est ensuite attribué à un véhicule ayant la capacité de le prendre en charge, construisant une solution initiale au problème comprenant déjà des éléments d'optimisation.

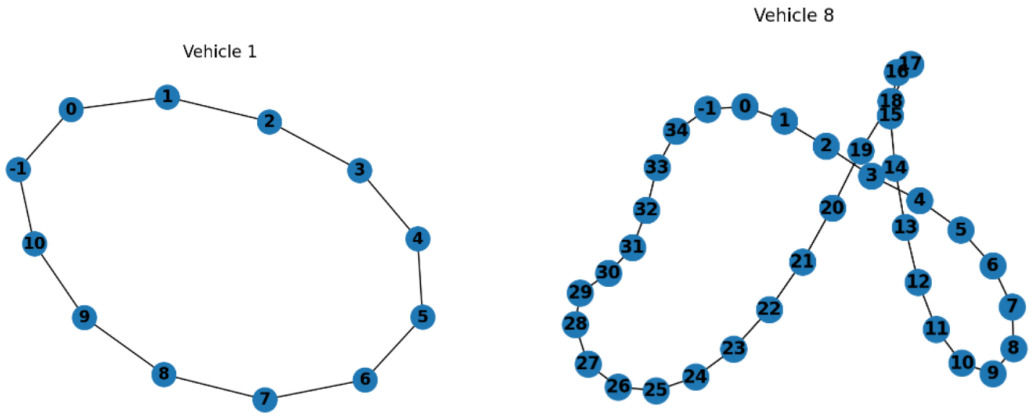
B. Visualization et Streamlit

A partir de notre base de données, il n'est pas évident de visualiser les données et de les comprendre. En effet, les données GPS de longitude et de latitude ne sont pas facilement compréhensibles par l'homme. C'est pourquoi nous avons pensé à afficher chaque client et dépôt dans une Leaflet/OpenStreeMap. Ainsi, on peut tout de suite comprendre plus clairement le problème posé et rassembler les points situés proches des autres. Pour cela, nous avons utilisé Streamlit, un framework de python capable de créer des sites web en-ligne pour visualisation et opération des applications de data.



Website créé avec Streamlit, avec les données des clients et dépôts affichées.

Mis à part les images déjà présentées ci-dessus (avec la carte interactive et le dénogramme de cluster), nous avons également créé un algorithme de visualisation simple créé avec networkx qui représente les clients dans les routes. Chaque véhicule commence par -1 (le Dépôt) et commence par ses clients 0 à N.



C. Optimisation du parcours de chaque véhicule

L'optimisation du parcours de chaque véhicule parmi ses clients a été attribuée aux algorithmes métaheuristiques, qui vont pendant un nombre d'itérations donné chercher à diminuer autant qu'ils le peuvent le coût du parcours. En mode système multi-agents (c'est à dire lorsqu'on utilise plusieurs métaheuristiques sur un même véhicule), ils vont de plus modifier leurs paramètres de recherche selon les résultats qu'ils ont obtenu et éventuellement échanger leur dernière solution en date selon le mode de fonctionnement fourni à l'agent évaluateur.

D. Répartition des clients entre les véhicules

Pour mieux explorer les solutions possibles et réduire les coûts, on peut aussi jouer sur la répartition des clients entre les véhicules afin de tester de nouvelles solutions initiales : c'est ce que fait le Q-Learning et son processus d'apprentissage. A partir de la première répartition obtenue avec l'attribution de clusters de clients aux véhicules, il teste diverses actions possibles sur une solution préalablement ordonnée par les métaheuristiques, en échangeant par exemple des clients entre les véhicules, puis il réordonne la solution et voit si la modification l'a améliorée ou pas. Il garde en mémoire les actions les plus efficaces et selon les coefficients exploration/exploitation il va les réitérer ou en essayer des nouvelles, jusqu'à ce qu'il n'arrive plus assez vite à améliorer la solution.

III. Optimisation par Métaheuristiques

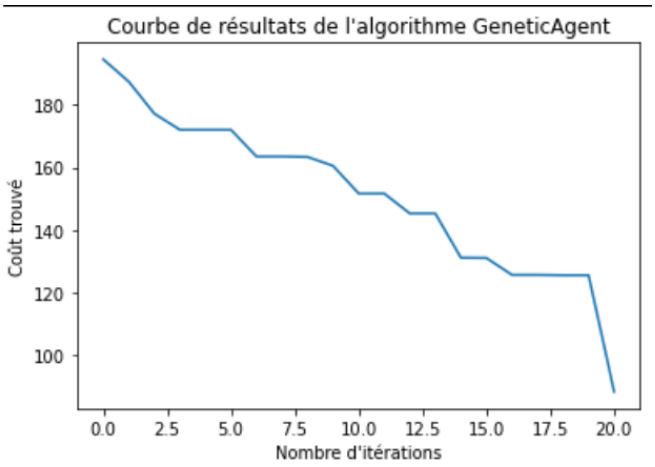
A. Tabou

1. Algorithmes spécifique PTVFT

```
Initialisation
        Choisir une solution admissible initiale s∈ X;
        s* := s;
        nbiter : = 0; (compteurs d'itérations)
        T := Ø; (la liste taboue est vide initialement)
        initialiser la fonction d'aspiration A;
        meil_iter := 0; (itération ayant conduit à la meilleure solution s* trouvée jusque-là)
Processus itératif
        tant que (f(s) > f˜) et (nbiter - meil_iter < nbmax) faire
                nbiter := nbiter + 1;
                générer un ensemble N' ⊆ N(s) de solutions voisines de s;
                choisir la meilleur solution s'∈N' telle que f(s') <= A(f(s)) ou s'∉T;
                mettre à jour la fonction d'aspiration A;
                mettre à jour la liste T des solutions taboues;
                s:= s ';
                si f(s) <f(s*) alors s*:=s; meil_iter:= nbiter;
        fin_tantque
fin de l'alg
```

Dans cet algorithme, l'état suivant est choisi de telle sorte qu'il minimise la fonction de coût sur le voisinage de l'état actuel. En dehors d'un optimum local, il agit comme une descente de gradient mais dans un optimum local, il choisit le moins mauvais voisin et continue sa recherche. Afin d'éviter de boucler sur des solutions déjà rencontrées, une liste de solutions "taboues" qu'on ne peut pas choisir est utilisée. Néanmoins, il est parfois préférable de repasser par une solution déjà visitée pour continuer la recherche de la solution optimale dans le voisinage de cette solution. On utilise alors une fonction d'aspiration qui permet de lever le caractère tabou d'une solution si elle est meilleure que l'ensemble des solutions rencontrées. L'arrêt de l'algorithme est réalisé lorsque la fonction objectif devient inférieure à une valeur fixée ou que le nombre d'itérations maximal est atteint.

2. Les courbes et tableaux



3. Analyse des résultats

Nous observons que l'algorithme génétique suit une diminution du coût par palier mais de manière régulière sauf à la fin où il y a une baisse importante. Cette baisse brutale rend le résultat final particulièrement bon. Le nombre d'itérations étant faible, l'algorithme est assez efficace.

4. Intérêt de l'étude

La méthode tabou est performante sur de nombreux problèmes d'optimisation. En revanche, dans le cas d'un minimum local "profond", l'algorithme va être très coûteux en espace mémoire. De plus, les conditions d'arrêt de l'algorithme ne sont pas optimales puisqu'il faut choisir préalablement la borne et le nombre d'itérations maximal et qu'un mauvais choix d'un de ces paramètres peut entraîner la non-convergence ou une convergence trop lente de la méthode tabou. Le choix de la solution initiale peut également poser problème dans certains cas et provoquer un allongement important du temps d'exécution de l'algorithme.

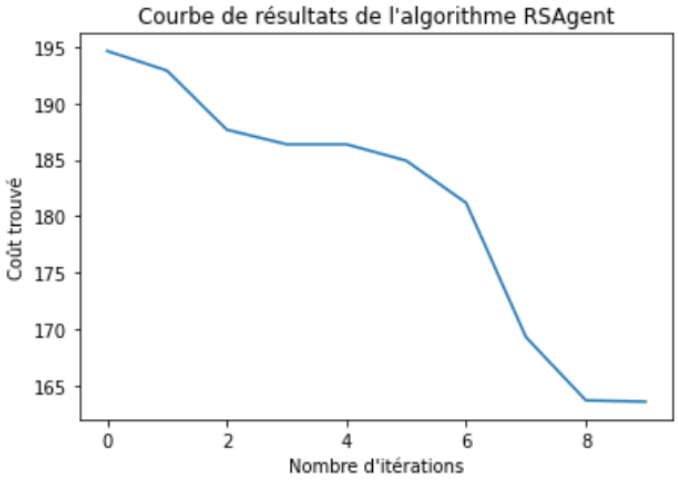
B. Recuit Simulé

1. Algorithmes spécifique PTVFT

```
Initialisation
        Choisir une solution admissible initiale s∈ X;
        s* := s;
        nbiter := 0; (compteur d'itération global)
        nouveau_cycle := vrai; (variable booléenne indiquant O/N effectuer un nouveau cycle d'itérations)
        t := t0; (t0 = température initiale du système)
Processus itératif
    tant que nouveau_cycle = vrai faire
        nbiter := 0; (compteur d'itération interne à un cycle)
        nouveau_cycle := faux;
        tant que nbiter < nbiter_cycle (nombre de paliers à fixer au départ) faire
            nbiter := nbiter + 1;
            générer une solution s' ∈ N(s) aléatoirement;
            δf := f(s') - f(s);
            si δf <0 alors s:= s'; nouveau_cycle := vrai;
            sinon prob(δf,t) := exp(- δf/t); générer q aléatoirement dans [0,1[;
                si q < prob(δf,t) alors s := s'; nouveau_cycle := vrai;
            si f(s) < f(s*) alors s* := s;
        fin_tantque
        t := a*t  (0<a<1 : coefficient de refroidissement)
    fin_tantque
fin de l 'alg
```

Le recuit simulé permet de simuler l'évolution d'un système instable vers un état d'équilibre. Le passage d'un état au suivant est effectué grâce à une perturbation du système. Si l'énergie de ce nouvel état est plus faible que l'énergie du précédent, le nouvel état remplace l'état précédent. Sinon, le nouvel état remplace le précédent avec une probabilité $exp(-\delta E/k_B T)$. Dans notre problème, les états correspondent à nos solutions, l'énergie correspond à notre fonction de coût et la température correspond à un paramètre qui permet de contrôler la convergence de l'algorithme. Chaque nouvelle solution proposée fait donc partie du voisinage de la solution actuelle. Avec cet algorithme, il est possible d'obtenir, temporairement, une solution dégradée afin d'en trouver une meilleure que la précédente (cela revient à s'extraire d'un minimum local afin de chercher le minimum global). La solution optimale est atteinte lorsqu'il n'y a plus de solution d'énergie plus faible dans le voisinage et que la température est suffisamment faible pour que même si l'énergie est plus élevée, la probabilité de remplacer l'état actuel est très proche de 0.

2. Les courbes et tableaux



3. Analyse des résultats

Nous observons une baisse importante lors des 7è et 8è solutions trouvées mais les précédentes n'ont pas permis de réduire significativement le coût.

4. Intérêt de l'étude

L'algorithme de recuit simulé, contrairement à d'autres métaheuristiques, converge vers un optimum global. La seule limite de cet algorithme est le temps nécessaire et donc le nombre d'itérations requises afin de converger vers la meilleure solution. Le recuit simulé peut converger dans un premier temps vers des optima locaux mais le fait d'autoriser de choisir un nouvel état d'énergie plus élevée avec une probabilité $exp(-\ \delta E/k_B T)$ permet d'en sortir et de pouvoir converger vers l'optimum global.

Néanmoins, le recuit simulé peut rester bloqué dans un optimum local dans le cas où la température atteindrait une valeur trop faible. Egalement, on ne peut pas savoir, avec le recuit simulé si la solution est optimale ou non et le choix de la température initiale peut avoir un impact important : si elle est trop basse, la recherche ne sera pas optimale et si elle est trop haute, le temps de recherche risque d'être trop important.

C. Algorithme génétique

1. Algorithmes spécifique PTVFT

⇒**Etape 0 : Définir un codage du problème**

⇒**Etape 1 : t:=0, créer une population initiale de N individus P(0) = x1,x2,...,xN**

=> Etape 2 : Evaluation

Calculer la force $F(x_i)$ de chaque individu xi, i=1…N

=> Etape 3 : Sélection

Sélectionner N individus de P(t) et les ranger dans un ensemble S(t). Un même individu de P(t) peut apparaître plusieurs fois dans S(t)

⇒**Etape 4 : Recombinaison**

Grouper les individus de S(t) par paire, puis, pour chaque paire d'individus :

- avec la probabilité P_{cross}, appliquer le croisement à la paire et recopier la progéniture dans S(t+1) (la paire d'individus est éliminée, elle est remplacée par sa progéniture),

- avec la probabilité 1-P_{cross}, recopier la paire d'individus dans S(t+1)

Pour chaque individu de S(t+1) :

- avec la probabilité P_{mut}, appliquer la mutation à l'individu le recopier dans P(t+1)

-avec la probabilité 1-P_{mut}, recopier l'individu dans P(t+1)

=> Etape 5 : Incrémenter *t* et reprendre à l'étape 2 jusqu'à un critère d'arrêt.

Les algorithmes génétiques sont basés sur l'évolution naturelle et la survie du plus fort. Afin de pouvoir utiliser ce type d'algorithmes, il faut représenter structurer la solution sous forme de chaînes. Le parallèle avec la génétique amène le vocabulaire utilisé à être celui de la biologie. L'idée de l'algorithme, après avoir généré une population initiale, est de sélectionner, croiser et muter une partie de la population dans le but de trouver de meilleures solutions.

L'algorithme fait appel à plusieurs opérateurs :

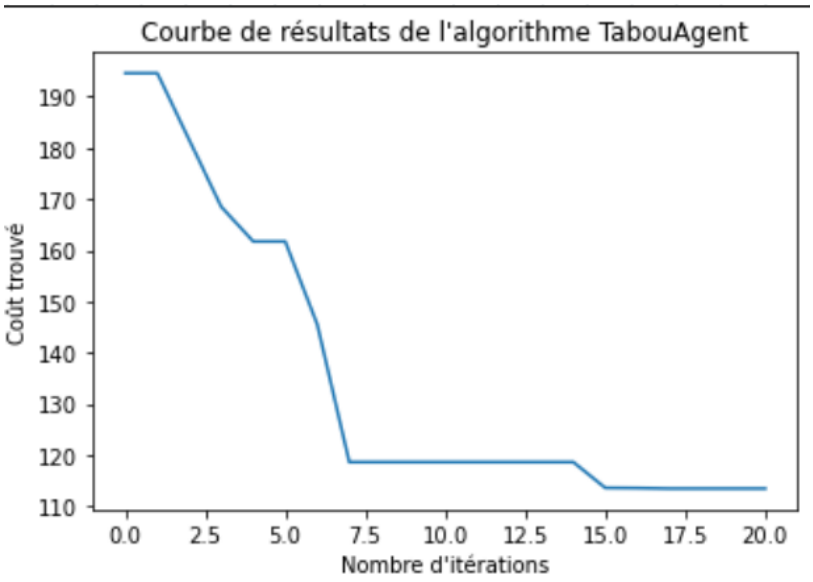
- Opérateur de sélection : il détermine quelle partie de la population survit grâce à une fonction d'évaluation qui permet de supprimer les solutions les moins performantes. Les parents ainsi sélectionnés sont alors croisés pour créer de nouvelles solutions ; cette étape peut être faite de plusieurs manières : croisement

à un point, deux points ou uniforme.

- Opérateur de mutation : il permet de changer le génotype d'une solution "fille" avec une faible probabilité. Il autorise des solutions moins performantes dans le but de garder la diversité de la population et ainsi trouver de meilleures solutions.

Le formalisme de l'algorithme est assuré par la représentation par blocs et son théorème.

2. Les courbes et tableaux



3. Analyse des résultats

Nous observons ici deux baisses importantes lors des premières itérations qui permettent d'aboutir très rapidement à une solution quasi-optimale que l'algorithme a ensuite du mal à améliorer. Nous pourrions donc diminuer le nombre d'itérations puisqu'elles ne sont pas toutes nécessaires à l'obtention d'une solution optimale.

4. Intérêt de l'étude

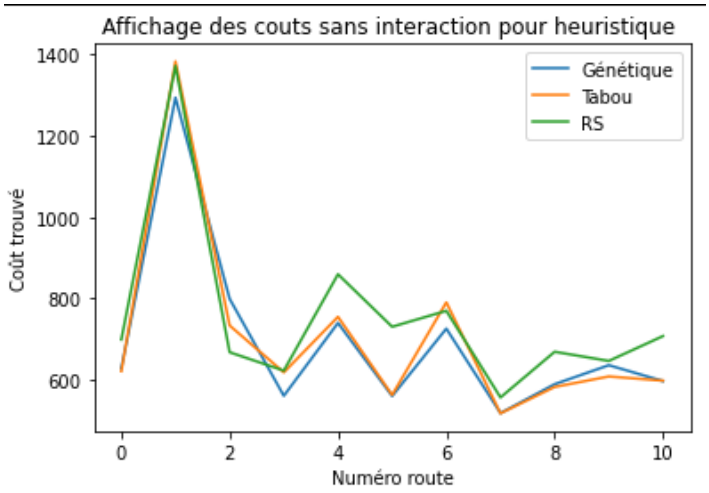
Les algorithmes génétiques ont moins de chances que les algorithmes classiques puisqu'ils explorent simultanément plusieurs solutions et le phénomène de mutation permet de conserver une diversité de solutions.

Le plus gros inconvénient des algorithmes génétiques réside dans la nécessité de

la structure de chaînes pour les solutions. Notre problème s'y prête bien mais ce n'est pas le cas de tous les problèmes d'optimisation.

Les algorithmes génétiques ne garantissent pas l'obtention de la solution optimale mais, dans les faits, la solution converge rapidement vers celle-ci.

D. Comparaison entre les trois heuristiques



On affiche ici le coût de la solution trouvée en fonction de la route. On observe que les trois algorithmes sont relativement proches malgré le fait que dans ce cas le Recuit Simulé trouve souvent une solution moins performante que les deux autres algorithmes. Les algorithmes tabou et génétique sont particulièrement proches sur l'ensemble des solutions trouvées avec les paramètres ici fixés.

Il convient aussi de remarquer que ce résultat est amené à varier de façon importante selon les paramètres en jeu, notamment pour ce qui est de la taille de la population de solutions pour les algorithmes génétiques et tabou, ou pour le nombre de cycles. En augmentant les dimensions de la population on augmente la mémoire nécessaire pour le fonctionnement de l'algorithme tout en donnant au final une meilleure solution.

V. Optimisation Collaborative : SMA+Métaheuristiques

En mode Système Multi Agents, on va associer à chaque agent véhicule des agents métaheuristiques. L'agent véhicule servira alors d'agent évaluateur, en centralisant les solutions renvoyées à chaque étape par les algorithmes métaheuristiques, en les

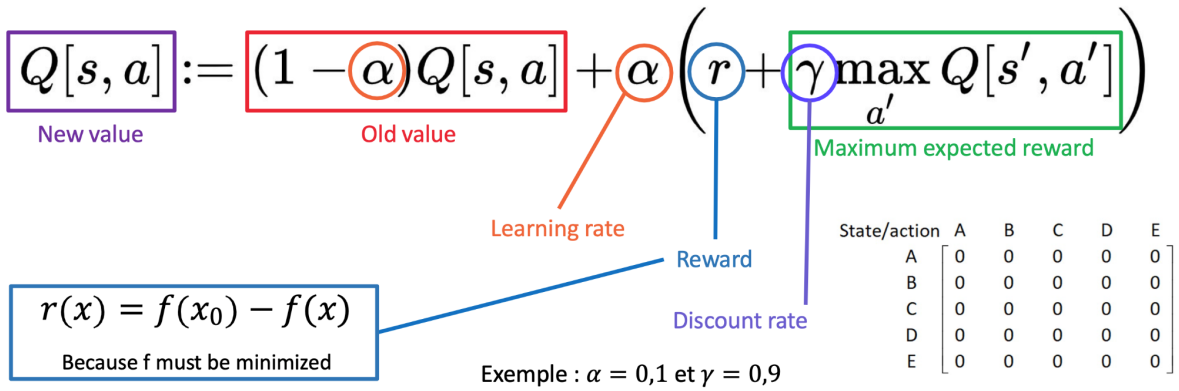
comparant et en choisissant selon la méthode d'interaction sélectionnée les informations qu'il leur fournit pour l'étape suivante. Nous avons décidé de développer trois méthodes d'interaction entre les agents métaheuristiques :

- En indépendance : chaque agent cherche la meilleure solution indépendamment, aucune information n'est envoyée aux agents métaheuristiques.
- En compétition : chaque agent cherche la meilleure solution indépendamment, seuls les paramètres de recherche sont mis à jour : les algorithmes n'ayant pas trouvé la meilleure solution sont améliorés en augmentant la taille de la population ou le nombre d'itérations dans le cas du recuit simulé.
- En collaboration : à chaque step, chaque agent cherche indépendamment la solution optimale puis les solutions sont mises en commun et les paramètres mis à jour avant de continuer les autres étapes avec la meilleure solution.

VI. Q Learning

Le Q Learning est un algorithme qui permet d'accélérer les recherches d'un état et d'une solution pour laquelle la récompense est maximale. On utilise le Q Learning afin d'améliorer la recherche de solution à ce problème. Cet algorithme est basé sur la fonction mathématique suivante:

Fonction d'utilité Q(s,a)



s représente l'état actuel du problème, s' l'état suivant que l'on atteint avec l'action a'. a est une action qui définit s, alpha est le learning rate appelé aussi facteur d'apprentissage et gamma est le discount rate appelé également facteur d'actualisation.

Le r est le reward, c'est-à-dire les récompenses obtenus à l'état s, qui sont dans notre problème le fait de diminuer le coût de la solution.

L'exploration est le fait de ne pas tenir compte des récompenses possibles et de simplement effectuer une action dont on calculera les récompenses possibles ensuite. L'exploitation est le cas inverse, c'est le fait de choisir l'action depuis un état s donné qui va maximiser les récompenses r de l'état s'. L'exploration et l'exploitation viennent de l'algorithme epsilon-greedy, qui est implémenté dans le Q Learning afin de choisir une action. Si l'on joue sur le facteur d'exploration/exploitation de cet algorithme, le Q Learning agira différemment selon la valeur, il décidera d'explorer de manière plus ou moins intensive et de ne pas tenir compte des récompenses, ou bien il n'explorera pas du tout, et agira seulement pour un gain de récompense, donc une baisse de coût de la solution.

La matrice Q est la matrice état/action dans laquelle on note toutes les récompenses possibles par paire état/action, l'algorithme choisira ensuite l'action disponible et maximisant la prochaine récompense, ou il décidera d'explorer d'autres actions.

L'algorithme Q Learning de notre problème contient 8 fonctions distinctes capables de modifier la solution, les fonctions sont:

1. Intra-Route Swap: échange d'un client avec un autre client dans la même route (les clients 4 et 6 de la route 2 sont échangés).
2. Inter-Route Swap: fonction de voisinage qui effectue le déplacement d'échange d'un client d'une route avec un client d'une autre route.

3. Intra-Route Shift: fonction de voisinage qui effectue le déplacement d'un client vers une autre position sur la même route.

4. Inter-Route Shift: fonction de voisinage qui effectue le déplacement d'un client d'une route à une autre.

5. Two Intra-Route Swap: fonction de voisinage qui consiste en l'échange de clients sur la même route, ainsi que la fonction de voisinage d'échange intra-route. Cependant, dans la fonction Two Intra-Route Swap, deux clients consécutifs sont échangés avec deux autres clients consécutifs de la même route;

6. Two Intra-Route Shift: fonction de voisinage qui consiste en la relocalisation des clients sur la même route, ainsi que la fonction de voisinage de shift intra-route. Cependant, dans la fonction Two Intra-Route Shift, deux clients consécutifs sont retirés de leur position et réinsérés dans une autre position de la même route;

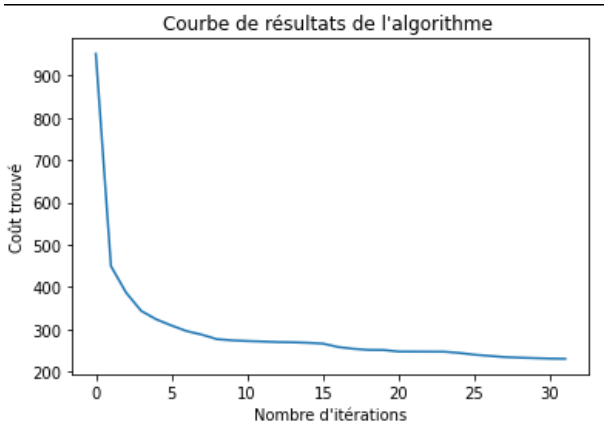
7. Élimine la plus petite route: fonction de voisinage qui cherche à éliminer la plus petite route de la solution.

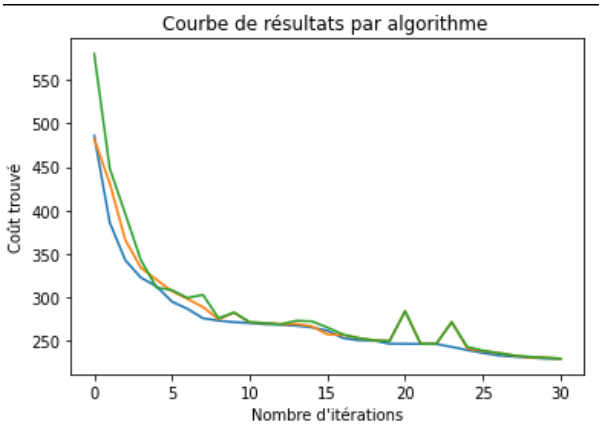
8. Élimine une route aléatoire: la fonction de voisinage Élimine la route aléatoire, fonctionne de la même manière que la fonction Élimine la route la plus petite, mais la route à supprimer est choisi au hasard

Le Q Learning pioche dans ces fonctions pour engendrer les actions qui changent l'état s en s', donc qui changent la solution afin de baisser son coût, qu'il s'agisse d'une modification transférant un client à un autre véhicule ou modifiant l'ordre dans la solution. Il agit en coopération avec les algorithmes Tabou, RS et génétique afin de les améliorer. On remarque une amélioration des résultats des algorithmes grâce au Q learning et son implémentation de la manière suivante:

```
1: procedure ADAPTIVELOCALSEARCHQLEARNIG(x_0)
2:     initialize(Q(state, action));
3:     improved ← true;
4:     no_improvement ← 0;
5:     x* ← x_0;
6:     x ← x_0;
7:     repeat                                          ▷ for each episode
8:         reward ← 0;
9:         states_visited_count ← 0;
10:        next_state ← chooseAnAction(0, 2);          ▷ 2: initial state
   defined by random function
11:        x ← bestNeighbor(next_state, x);
12:        if (x is better than x*) then
13:            x* ← x;
14:            reward ← x.getFitnessLearning();
15:        else
16:            states_visited_count + +;
17:            while (not reached the state goal) do   ▷ state goal:
   improving the solution
18:                if (no_improvement = 0) then
19:                    state ← next_state;
20:                    next_state = chooseAnAction(state, 1);   ▷ 1:
   epsilon greedy function
21:                else
22:                    next_state = chooseAnAction(0, 2);       ▷ if not
   improved, the greedy function should not be used
23:                end if
24:                x = bestNeighbor(next_state, x);
25:                if (x is better than x*) then    ▷ reached the state
   goal
26:                    x* ← x;
27:                    improved ← true;
28:                    no_improvement ← 0;
29:                    reward ← reward + x.getFitnessLearning();
30:                    calculateQValue(state, next_state, reward);
31:                else
32:                    no_improvement + +;
33:                    if (the state has been visited) then
34:                        states_visited_count + +;
35:                    end if
36:                    if ((no_improvement > max_iterations_without_improvement) and (states_visited_count = q_size)) then
37:                        improved ← false;
38:                    end if
39:                end if
40:            end while
41:            ϵ ← ϵ ∗ decay_rate;
42:        end if
43:    until (not(improved))
44:    return x;
45: end procedure
```

La condition d'arrêt du Q Learning est obtenue lorsque celui-ci n'a pas réussi à améliorer la solution pendant un certain nombre de tours, solution qui arrivera forcément car la décroissance du coût suit une hyperbole :



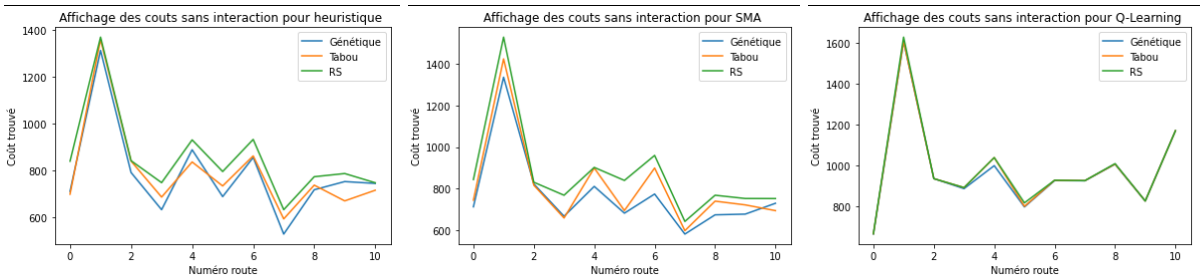


IX. Tableaux de comparaison et analyses globales des performances de l'optimisation collaborative

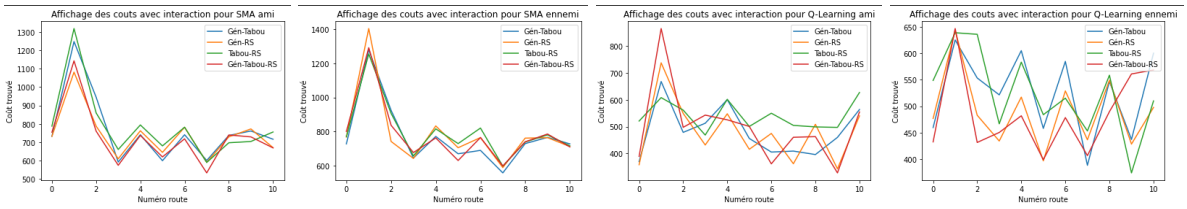
Nous avons par la suite réalisé deux expériences dans le but d'analyser l'efficacité des algorithmes, une dans une situation avec comparaison et l'autre dans une situation sans comparaison. Les paramètres choisis étaient les suivants :

```
nb_ite = 25
route_num = len(model.agents['routes'])
nb_algs = 3

#Q-Learning params
max_iter_no_improvement = 10
max_nb_states = 8
epsilon = 0.8
decay_rate = 0.8
learn_rate = 0.1
disc_rate = 0.9

#Heuristic params
pcross = 0.2
pmut = 0.5
taille_pop = 10
iter_cycle = 10
refroidissement = 0.95
w = 10
```

Résultats obtenus avec indépendance des agents: dans le cas de l'indépendance des agents, on voit qu'il y a peu d'écart de coût entre les algorithmes heuristiques isolés, les algorithmes en système multi-agents et les algorithmes en Q-Learning. Ceci s'explique par le fait que sans interaction il n'y a pas de paramètres permettant l'amélioration de la recherche mis en commun, et donc le comportement des algorithmes ne change pas, ce qui explique pourquoi les solutions trouvées ont des coûts si proches. On peut toutefois s'intéresser aux écarts de valeurs entre chaque algorithme : en effet, on remarque une influence décisive des paramètres sur le coût obtenu, avec dans les deux premiers cas le coût le plus faible obtenu respectivement par l'algorithme génétique, puis tabou, puis recuit simulé. Cet écart est effacé dans le Q-Learning par la mise à jour de la solution après chaque action.



Résultats obtenus avec interaction entre agents : Contrairement à ce qu'on a pu voir précédemment, les coûts obtenus lorsque les algorithmes sont mis en interaction diminue drastiquement entre le système multi-agents et le Q-Learning, pour lequel les valeurs ont souvent été divisés par deux par rapport au SMA. Ceci atteste de l'efficacité de l'apprentissage, mais aussi du mode d'interaction. En effet, on constate que les coûts obtenus dans le mode ami sont légèrement plus faibles que ceux obtenus dans le mode ennemi dans la plupart des cas.



Pour ce qui est de la comparaison selon les combinaisons d'algorithmes heuristiques choisies, on observe dans le cas du SMA que si certaines combinaisons sont avec les paramètres opérationnels choisis généralement plus efficaces que d'autres en matière de coût, les coûts obtenus restent assez proches avec pour meilleure solution la combinaison des trois algorithmes. Cette comparaison ne peut pas vraiment être faite

dans le cas du Q-Learning, dans la mesure où chaque combinaison d'algorithme travaillera avec des solutions issues d'actions différentes dans chaque situation et pour chaque route, et que nous avons mis des critères stricts sur la recherche de solutions faisant que l'on aboutit pas toujours à la meilleure. On peut quand même remarquer cependant que certaines solutions font consensus.

VII. Conclusions

Avec ce projet, nous avons beaucoup appris sur l'optimisation, non seulement en ce qui concerne les algorithmes heuristiques typiques, mais aussi sur de nombreuses façons innovantes de les relier dans un système multi-agent avec ou sans Q-Learning. Cela impliquait notamment de comprendre comment construire un système complexe à partir d'algorithmes de base et de voir comment ils étaient liés les uns aux autres.

Nous avons appliqué avec succès les algorithmes qui nous ont été enseignés, même si leurs performances pouvaient encore être améliorées, notamment dans la partie communication entre algorithmes. Ces défauts dans la conception ont toutefois pu être palliés par des solutions astucieuses telles que la clusterisation, la cartographie et le tracé des itinéraires des véhicules.

L'une de nos idées originales était d'essayer d'aller directement à l'idée 100 % POO/SMA, qui non seulement nous à aidé à moduler le problème, mais nous a aussi permis de jeter les bases de nombreuses idées intéressantes, comme celle de donner aux clients et aux dépôts quelque chose à faire entre les étapes : Par exemple, dans une future simulation, nous pourrions avoir les clients susceptibles de demander de nouveaux articles

Ce projet a aussi été une excellente occasion de travailler sur un problème réel que de nombreuses entreprises tentent de résoudre, en utilisant des données réelles. Nous avons de surcroît fait de notre mieux pour prendre en compte tous les problèmes et limitations apparaissant dans les jeux de données. Nous sommes satisfaits de nos résultats, sachant que nous avons maintenant appris et que nous pouvons appliquer nos compétences à des problèmes encore plus difficiles.