Sommaire :

1. Problématique apportée par le sujet
   1. Contrainte de temps
   2. Contrainte de coûts
2. L’implémentation
   1. Structures de données
3. L’heuristique
   1. Construction des batches
   2. Test des permutations de batches
4. Résolution exacte
   1. Construction des batches
   2. Test de de toutes les possibilités avec cut
   3. Optimisation : tri de la liste initiale
5. Améliorations possibles

# Problématique amenée par le sujet

# Implémentation

Nous avons choisi d’implémenter notre algorithme grâce à un langage orienté objet, qui nous permettait de représenter plutôt intuitivement les paramètres de l’énoncé. Entre Java et C++, nous nous sommes penchés sur C++, qui offre une plus grande liberté d’implémentation. Après quelques recherches, la performance entre ces deux langages n’est pas forcément évidente. Nous avons un petit regret pour notre choix, puisque C++, de par son exigence, nous a parfois joué des tours.

Voici les structures de données présentes dans le programme :

Probleme :

C’est la classe principale du programme. C’est cette classe qui regroupe toutes les données du problème.

Attributs :

* Capa : la capacité maximum du transporteur
* Eta : le coefficient de coût d’un trajet
* Clients : la liste de tous les clients du problème
* Produits : la liste de tous les jobs du problème
* bestSol : la meilleure solution trouvée au problème. Au début, il n’y en a pas. Une solution est une liste de Batches. Leur ordre dans la liste est l’ordre dans lequel il faut les envoyer.
* evalBestSol : l’évaluation de la meilleure solution trouvée

Méthodes :

* heuristique : créé une solution heuristique
* solve : trouve la solution exacte au problème

Il y a beaucoup d’autres méthodes que nous n’énumèrerons pas, puisqu’elles représentent des sous-méthodes nécessaires au bon fonctionnement de « heuristique » et de « solve ».

Nous ne parlerons pas de la classe Parser, chargée de découper le fichier de données, qui ne constitue qu’un outil inutile à la résolution.