Compte rendu du TP3 – Problème de Tournées de Véhicules (VRP)

Rayan SAKHY
ISIMA – École d'Ingénieurs en Informatique

Décembre 2025

Résumé

Ce compte rendu présente la conception, l'implémentation et l'évaluation d'une chaîne de résolution pour le **VRP** (Vehicle Routing Problem) dans le cadre du TP3 de Recherche Opérationnelle. Nous détaillons : (i) trois heuristiques de construction (Plus Proche Voisin, version randomisée, et une heuristique libre), (ii) la **méthode de split** (flotte illimitée et limitée) qui transforme une tournée TSP géante en solution VRP faisable, et (iii) une **recherche locale** (2-OPT, Relocate, Swap) orchestrée par un schéma *VND*. Nous incluons enfin un retour d'expérience sur les instances fournies et les correctifs appliqués.

Table des matières

1 Contexte et objectifs **3**

2 Format des données et instances **3**
2.1 Instance pédagogique (ExCours) . . . 3
2.2 Instance générale (Ex.txt) . . . 3

3 Heuristiques de construction **3**
3.1 Plus Proche Voisin . . . 3
3.2 Plus Proche Voisin randomisé . . . 3
3.3 Heuristique libre ("Patate") . . . 3

4 Méthode de split **4**
4.1 Principe commun . . . 4
4.2 Split illimité (DP 1D) . . . 4
4.3 Split limité (DP 2D) . . . 4
4.4 Choix de l'infini numérique . . . 4

5 Recherche locale et VND **5**
5.1 2-OPT intra-tournée . . . 5
5.2 Relocate . . . 5
5.3 Swap inter-tournées . . . 5
5.4 VND (Variable Neighborhood Descent) . . . 5

6 Expérimentations et retours **5**
6.1 Pipeline d'exécution 5
6.2 Anomalies observées et correctifs 6

7 Expérimentations et résultats **6**
7.1 Instance grande (173 villes) 6
7.2 Split illimité vs limité 6
7.3 Amélioration par GRASP 6
7.4 Exemple de tournées après GRASP (cas limité) 6

8 Conclusion et perspectives **6**

A Annexe A – Spécification des fichiers **7**

B Annexe B – Paramétrage de la VND **7**

1 Contexte et objectifs

Le sujet du TP3 demande de récupérer des instances VRP, d'implanter trois heuristiques de construction d'une solution initiale, de réaliser une *recherche locale* avec mouvements 2-OPT et déplacement de sommet, et de construire une *méthode de split* pour passer d'une tournée TSP à des tournées VRP faisables. Une métaheuristique (GRASP, génétique/mémétique) est également suggérée, et deux cas doivent être considérés : flotte illimitée et flotte limitée. [1]

2 Format des données et instances

2.1 Instance pédagogique (ExCours)

L'instance d'exemple comporte 5 villes (clients) et une flotte de 2 véhicules de capacité 10. Les distances sont données dans une matrice incluant le dépôt (indice 0) et les clients (1..5), suivie des quantités demandées par ville. [2]

2.2 Instance générale (Ex.txt)

Une instance plus grande (nb clients = 173, nb camions = 30, capacité = 1220) a été utilisée pour valider l'échelle et la robustesse des algorithmes. Les demandes sont toutes $\leq$ 1220, ce qui garantit qu'un client peut toujours être servi seul si nécessaire. La matrice des distances est dense et contient des valeurs de l'ordre de 10^4 à 10^5, ce qui influe sur le choix d'une constante d'infini numérique appropriée dans la DP de split.

3 Heuristiques de construction

3.1 Plus Proche Voisin

Le Plus Proche Voisin construit un ordre TSP (*tour géant*) en ajoutant à chaque étape le client non encore visité le plus proche du dernier inséré. L'implémentation parcourt les clients restants et choisit le min de la ligne correspondante dans la matrice de distances.

3.2 Plus Proche Voisin randomisé

La version randomisée conserve le principe du Plus Proche Voisin mais sélectionne aléatoirement un client parmi une liste (RCL) des k plus proches. Cela permet d'explorer des ordres voisins du Plus Proche Voisin déterministe tout en introduisant de la diversité, utile pour les métaheuristiques (GRASP).

3.3 Heuristique libre ("Patate")

Une heuristique supplémentaire combine un tri des candidats par distance et une alternance proche/éloigné pour créer une tournée TSP moins biaisée. Cette idée favorise une répartition plus homogène des segments à l'étape de split.

1. Extrait du sujet "+TP (noté) numéro 3 : VRP+" et du document d'accompagnement (lien vers les instances HVRP) : `http://fc.isima.fr/~lacomme/hvrp/hvrp.html`.

2. Voir le fichier `ExCours.txt` : première ligne = nb villes, deuxième ligne = nb camions et capacité, puis matrice de distances et vecteur des demandes.

4 Méthode de split

4.1 Principe commun

On considère la tournée TSP géante ORD$[1..n]$ (le dépôt n'est pas inclus). On définit un **graphe acyclique** de positions $0..n$, où un arc $(i \to j)$ représente une tournée VRP qui dessert les clients ORD$[i+1], \ldots,$ ORD$[j]$ en respectant la capacité Q. Le coût de l'arc est :

$$c(i,j) = d(0, \text{ORD}[i+1]) + \sum_{k=i+1}^{j-1} d(\text{ORD}[k], \text{ORD}[k+1]) + d(\text{ORD}[j], 0).$$

La **DP** 1D (flotte illimitée) ou 2D (flotte limitée) recherche le plus court chemin de 0 à n (ou avec K tournées maximum).

4.2 Split illimité (DP 1D)

Algorithm 1 Split illimité (flotte non bornée)

Calculer les préfixes de demandes PREF pour tester vol$(i..j) \le Q$ en $O(1)$.
for $i = 1..n$ **do**
  $c_{\text{chain}} \leftarrow 0$; $c_{\text{df}} \leftarrow d(0, \text{ORD}[i])$.
  for $j = i..n$ **do**
    si PREF$[j]$ − PREF$[i-1] > Q$ **alors break**
    si $j > i$ **alors** c_{chain} += $d(\text{ORD}[j-1], \text{ORD}[j])$.
    COST$[i][j] \leftarrow c_{\text{df}} + c_{\text{chain}} + d(\text{ORD}[j], 0)$.
  end for
end for
DP$[0] \leftarrow 0$; DP$[j] \leftarrow +\infty$.
for $j = 1..n$ **do**
  for $i = 1..j$ **do**
    if COST$[i][j] \neq +\infty$ **then**
      DP$[j] \leftarrow \min\{\text{DP}[j],\ \text{DP}[i-1] + \text{COST}[i][j]\}$.
    end if
  end for
end for
Reconstruire les coupures via `parent[j]`.

4.3 Split limité (DP 2D)

On ajoute une dimension k pour le nombre de tournées utilisées :

$$\text{DP}[k][j] = \min_{1 \le i \le j} \big(\text{DP}[k-1][i-1] + \text{COST}[i][j]\big),$$

avec reconstruction par `parent[k][j]`.

4.4 Choix de l'infini numérique

Point d'attention majeur. Sur des matrices de distance avec des valeurs élevées (10^4 à 10^5), initialiser la DP avec INF = 10000 est **incorrect**. Il faut utiliser $+\infty$ (ou une borne réellement

supérieure aux coûts potentiels), sinon la DP refuse des transitions valides et la reconstruction échoue (`parent[j] = -1`). Dans notre code, le correctif consiste à remplacer `constexpr int INF = 10000;` par `const double INF = std::numeric_limits<double>::infinity();` dans les deux split et à conserver tous les coûts en `double`.

5 Recherche locale et VND

5.1 2-OPT intra-tournée

On remplace deux arêtes (a, u) et (v, b) par (a, v) et (u, b). Si le Δ-coût est positif, on inverse le segment $[i..j]$ et on *recalcule* le coût exact de la tournée.

5.2 Relocate

On supprime un client u de (R_1, i) et on l'insère dans (R_2, j) en respectant la capacité. **Point critique** : ne jamais décrémenter `taille` avant d'accepter le mouvement ; ajuster l'indice d'insertion $j \to j\,j$ si $R_1 = R_2$; puis *recalculer* coût et volume, et supprimer toute tournée vide.

5.3 Swap inter-tournées

On échange deux clients $u \in R_1$ et $v \in R_2$ en vérifiant la capacité des deux tournées. On met à jour les coûts par *recalcul exact* et on normalise la solution.

5.4 VND (Variable Neighborhood Descent)

Le VND applique cycliquement 2-OPT, Relocate, puis Swap, et **redémarre au premier voisinage** dès qu'une amélioration est trouvée. Des limites (itérations, temps, stagnation) peuvent être ajoutées pour éviter les boucles.

6 Expérimentations et retours

6.1 Pipeline d'exécution

La fonction `main` lit l'instance, construit des ordres TSP (Plus Proche Voisin et Plus Proche Voisin randomisé), effectue le split illimité, puis lance la VND et affiche la solution.

Listing 1 – Extrait de `Tp3.cpp` : pipeline simplifié

```
int main(){
  srand(42);
  t_probleme prob; t_solution sol;
  lire_instance("Ex.txt", prob);
  plusProcheVoisin(prob, sol);
  plusProcheVoisinRandom(sol, prob);
  if (splitVRPunlimited(prob, sol)) afficherTournees(sol);
  vndImprove(sol, prob);
  afficherTournees(sol);
}
```

6.2 Anomalies observées et correctifs

— **Coûts négatifs ou tournées** $0 \to 0$: ils proviennent d'une mise à jour incrémentale incohérente ou d'une suppression/ insertion mal indexée. Correctif : *recalcul* systématique des coûts et volumes après tout mouvement, puis `normalizeSolution`.

— **Split qui échoue** sur grande instance : dû à `INF=10000`. Correctif : passer à $+\infty$ en `double` dans la DP.

— **Couverture des clients** : un `checkCoverage` vérifie que chaque client apparaît exactement une fois ; en cas d'échec, un *re-split* de secours reconstruit la solution depuis l'ordre courant.

7 Expérimentations et résultats

7.1 Instance grande (173 villes)

— **Paramètres** : $n = 173$, $K = 30$, capacité $Q = 1220$.

— **Vecteurs TSP générés** :

— Plus Proche Voisin

— Plus Proche Voisin Randomisé

— Heuristique libre (“patate”)

7.2 Split illimité vs limité

Méthode	Nb tournées	Coût total
Split illimité	33	9 442 942
Split limité ($K = 30$)	30	9 447 188

7.3 Amélioration par GRASP

Cas	Nb tournées	Coût total
Illimité	33	5 735 173
Limité ($K = 30$)	30	5 692 557

7.4 Exemple de tournées après GRASP (cas limité)

```
T1 (vol=645, cout=57661): 0 -> 40 -> 4 -> 13 -> 137 -> 128 -> 22 -> 76 -> 0
T5 (vol=1207, cout=224240): 0 -> 77 -> 133 -> 107 -> 118 -> 62 -> 74 -> ...
T30 (vol=1082, cout=177041): 0 -> 163 -> 146 -> 91 -> 45 -> 64 -> 46 -> ...
```

On observe une réduction significative du coût total (40%) grâce à la diversification des ordres TSP et à la recherche locale intégrée dans GRASP. s

8 Conclusion et perspectives

Nous avons implémenté une chaîne VRP complète : heuristiques de construction, méthode de split (illimité et limité) et recherche locale (2-OPT, Relocate, Swap) orchestrée par VND. Les principaux enseignements sont : (i) utiliser un *infini* numérique adapté dans la DP de split, (ii)

recalculer les coûts après chaque mouvement pour garantir la cohérence, et (iii) *normaliser* la solution pour supprimer toute tournée vide. En perspective, l'intégration d'un **GRASP** complet, l'ajout de mouvements *Relocate k-chaînes* et *Swap de segments*, ainsi qu'une ILS (Iterated Local Search) permettraient d'améliorer la qualité des solutions sur les grandes instances.

A Annexe A – Spécification des fichiers

— `ExCours.txt` : petite instance pédagogique (5 villes, 2 camions, capacité 10, matrice des distances, demandes).

— `Ex.txt` : instance grande (173 villes, 30 camions, capacité 1220, matrice dense).

— `Source.h / Source.cpp` : définitions des structures, lecture d'instance, heuristiques, split, VND, et utilitaires d'affichage.

— `Tp3.cpp` : point d'entrée, pipeline d'exécution.

B Annexe B – Paramétrage de la VND

— `maxVNDIter = 2000`, `maxOptIter = 200`, `maxMoves = 5000`, `maxSwap = 3000`, `stallLimit = 300`.