

TP3

VRP avec fenêtre de temps

(minimiser le nombre de routes PUIS la distance totale
 instances de Solomon → taille 10)
 $\text{DR}_\text{opt} \rightarrow$ meilleurs résultats de la littérature

`data.h/cpp` → DD pour sommet → Customer

arcs → Arc

graph → Data

dans Data modes - sommet

arcs - arcs

distances - matrice de distance

`data.h/cpp` ligne le meilleur résultat sur l'instance trouvée

working solution, h/cpp

500 pour la solution courante

creer solution

durée : $i \in \mathbb{Z}$

vider

ouvrir l'ournée vide

ouvrir l'ournée avec 1 client

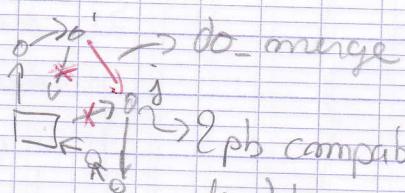
embarquer un client

échapper ce client

vérifier la validité de la solution (check)

`algorithms.h/cpp`

dummy →



+ capacité

$\rightarrow \text{schad}(\text{tour}(i)) + \text{dmd}(\text{tour}(i)) / KC$

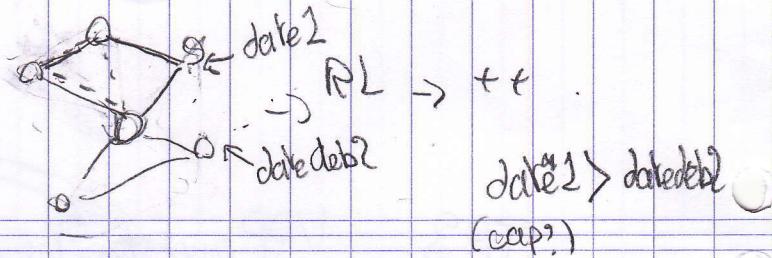
\rightarrow calculer date arrivée suiv en partant de i

→ vérifier sur TW[i]

heuristiques constructives

insertion

fusion



heuristique
insertion

insertion:

creer la tournée vide

faire

recherche client libre dont l'insertion

- est valide \rightarrow possibilité, cap non sollicité, fenêtre de temps

- coûte le moins cher

s'il existe
l'insérer

sinon

ouvrir une nouvelle tournée
tant qu'il existe client libre

Fusion

trier tous les arcs (i,j) selon le gain $g_{i,j} = c_{i,o} + c_{o,j} - c_{i,j}$ décroissant
créer la tournée "dummy" + arc (i,j) dans l'ordre

- regarder si correspond à une fusion potentielle
- si oui vérifier réalisable (capa, TW)
- si oui faire la fusion

distance
 γ = valeur de l'arc

\forall arc (i,j) calcul gain $g_{i,j} = e_{i,o} + e_{o,j} - e_{i,j}$

trier les arcs selon $g_{i,j}$

\Rightarrow pour arc $(i,j) \in$ ordre

| tester si (i,j) est tq $i = \text{fin}, j = \text{début}$

Si oui

vérifier respect capa

Si Oui

vérifier respect TW

Si Oui

fusionner

heuristique
fusion



Insertion

On pour d'une solution vide

ouvrir une tournée vide

Tant qu'il existe client non traité

rechercher le client non traité dont l'insertion
à la fin de la tournée courante est possible
et coûte le moins cher

Si le client existe

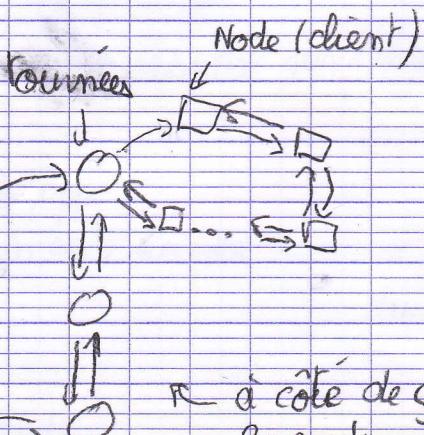
L'insérer à la fin de la tournée courante

Sinon

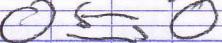
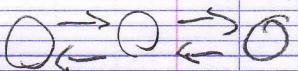
On ouvre une nouvelle tournée vide

Class Working solution:

Depot first
free last



→ à côté de ça
les routes sont stockées
sur un vector.



Tournées libres

dummy → pour fusion

Recherche d'insertion fusion

recherche locale

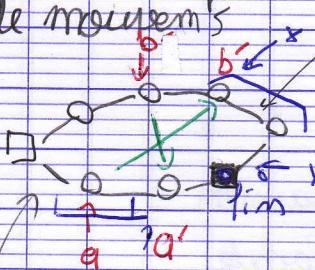


Type de mouvements

⑧ $2opt^*$



1 tournée
1 autre tournée



$\forall a \in S$

$\forall b \in S$

(if $b = \text{dépôt}$)

Reste mouve $2opt^*(a, b)$

change =
 $y - x$

→ vérifier gain en distance

→ vérifier capacité

→ vérifier TW { a'

{ b' }

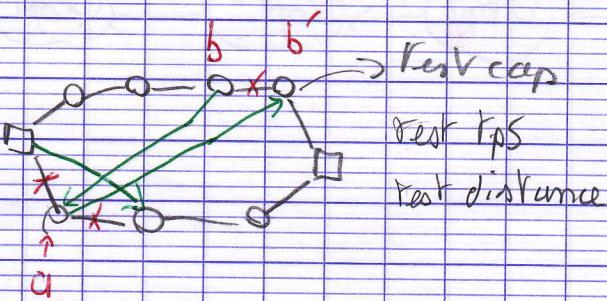
if $b = \text{dépôt}$

cas particulier = mettre la 2ème tournée à la fin (si possible)

- ⑨ on ne considère plus tous les (a, b) sur les tournées
- 2 dans la même fenêtre de temps

⑩ $0opt$

Extraire un sommet pour le réinsérer ailleurs

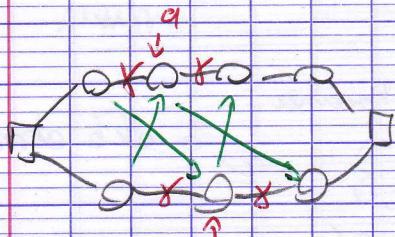


Cas particulier

- a seul client → suppression de la tournée
- a et b m tournée → recalcul des temps de passage
- a et b voyage → recup (calcul simplifié)
- 2 dans la même fenêtre de temps aussi

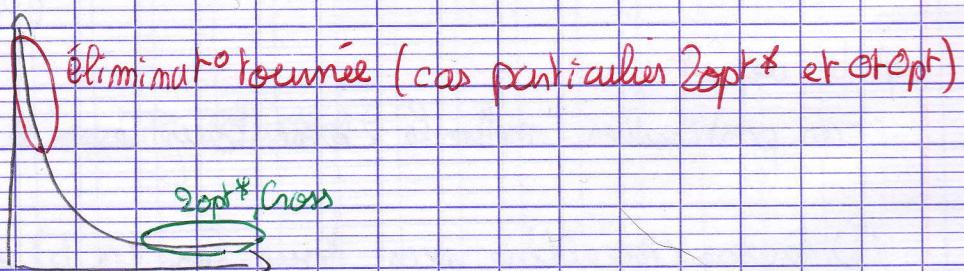
⑧ Cross (échange)

échange 2 clients



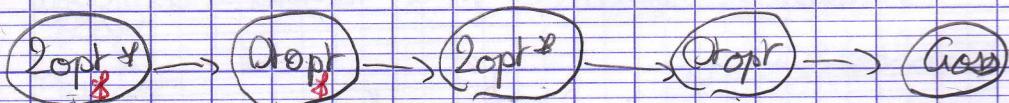
↳ ne réduit pas le nombre de tournées
Or il en fin de recherche locale

Recherche locale allure



Etapes RL

Passage au suivant qd plus d'amélioration, retour qd amélioré



* échecs partiels

VND

$$s \leftarrow s_0$$

$$k \leftarrow 1$$

faire

s' améliorante $\in V_k \rightarrow$

si s' exister

$$s \leftarrow s'; k \leftarrow 1;$$

sinon $k \leftarrow k+1$

tant que $k < b$

return s