PLD Agile

Partie 5 : Présentation du Projet Longue Durée

Christine Solnon

INSA de Lyon - 4IF

2018 / 2019

Plan de la partie 5

Présentation du Projet Longue Durée

Le projet Optimod'Lyon

- Réponse à un appel à projets de l'ADEME sur la mobilité urbaine
- Porteur : Grand Lyon
- 13 partenaires :
  - 2 collectivités : Lyon et Grand Lyon
  - 8 industriels : IBM, Renault Trucks, Orange, CityWay, Phoenix ISI, Parkeon, Autoroutes Trafic, Geoloc Systems
  - 3 laboratoires de recherche : LIRIS, CETE, LET
- Durée : 3 ans (2012-2015)
- Budget : 7 Millions

Objectifs d'Optimod'Lyon

nouveaux systèmes de collecte temps réel de données mobilités

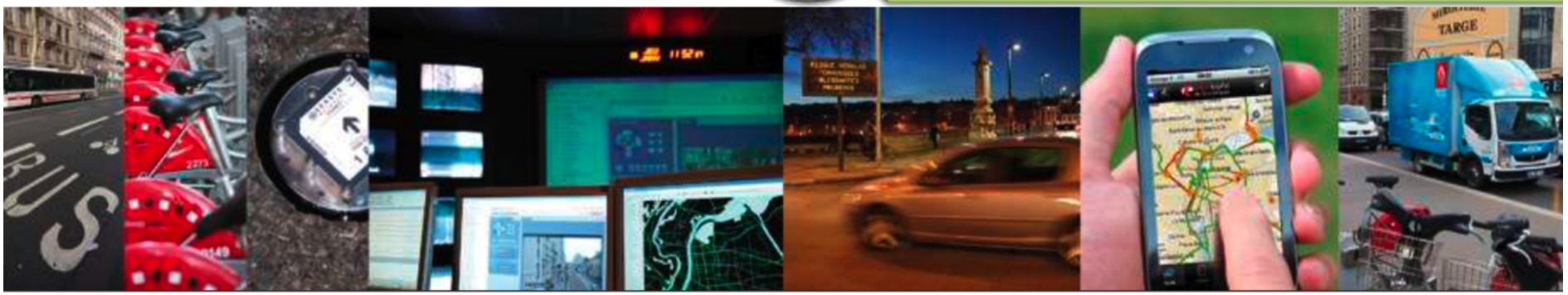
Portail mobilité du Grand Lyon

Entrepôt des données mobilité du territoire

L'optimisation de l'exploitation des réseaux urbains par la prédiction à 1h du trafic

La fourniture d'une information tous modes, temps réel, disponible à tout moment, en tout lieu et pour tous

L'optimisation de la gestion du fret urbain par l'information des conducteurs et la gestion des tournées des opérateurs



Plan de la partie 5

Présentation du Projet Longue Durée

Description de l'application

Votre mission :

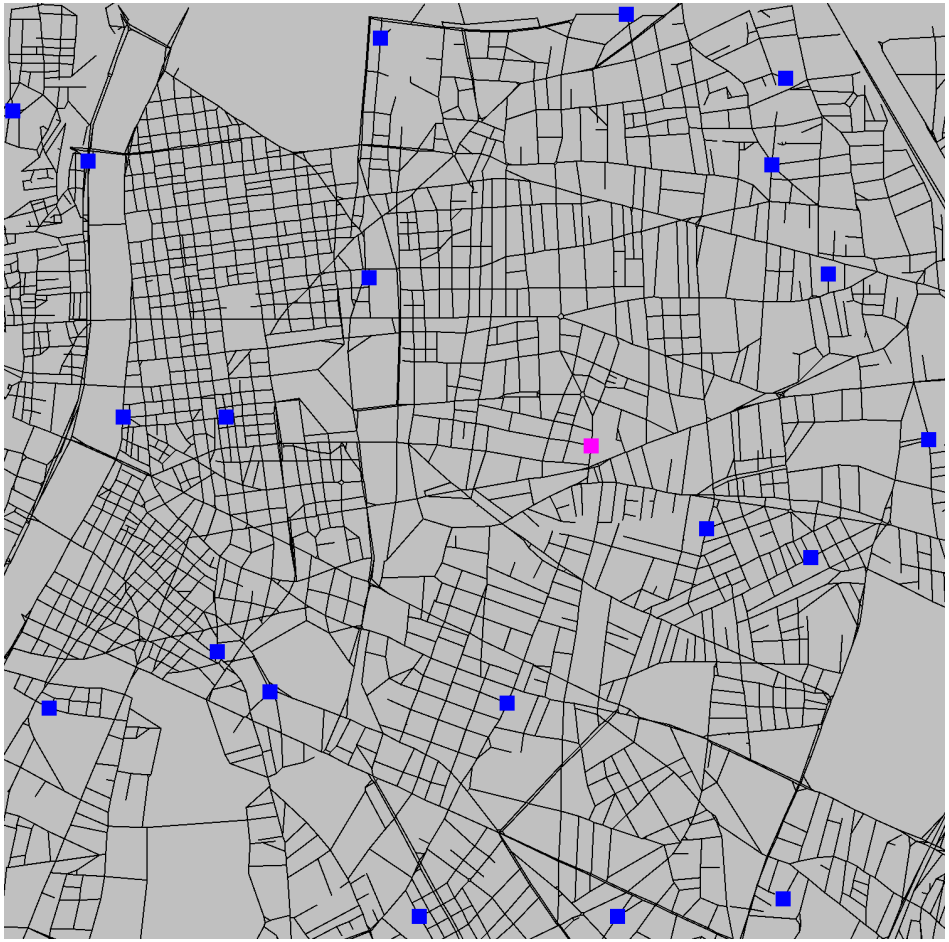
- Concevoir une application pour permettre à des sociétés de livraison de préparer leurs tournées
- Nouveauté 2018 : les livreurs sont à vélo !
  ⇝ Il y a plusieurs livreurs
  ⇝ Chaque livreur doit livrer le même nombre de colis (à 1 près)

Cas d'utilisation

- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer les tournées pour une demande de livraisons
- Modifier interactivement les tournées

Plan = Un ensemble d'intersections reliées par des tronçons
$\rightsquigarrow$ Chaque tronçon a une longueur et un nom de rue

Cas d'utilisation

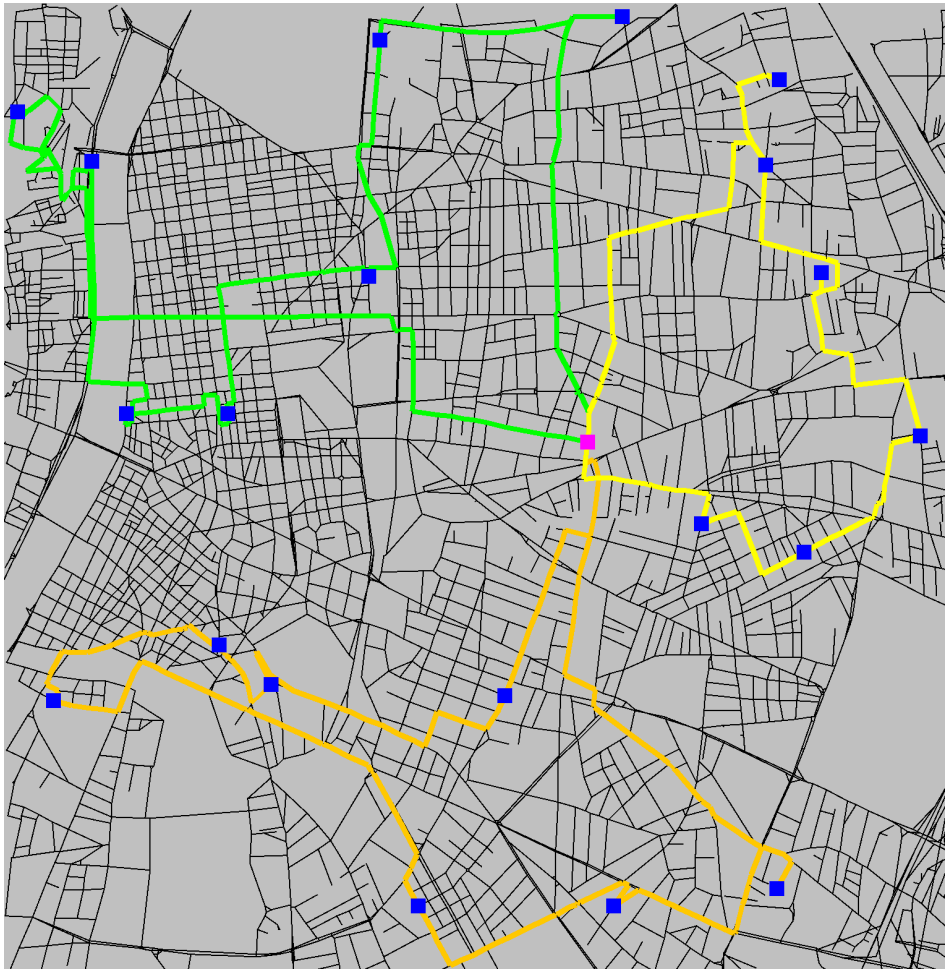


- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer les tournées pour une demande de livraisons
- Modifier interactivement les tournées

Demande de livraisons =

- Un entrepôt (point rose) et une heure de départ
- Des livraisons (points bleus)

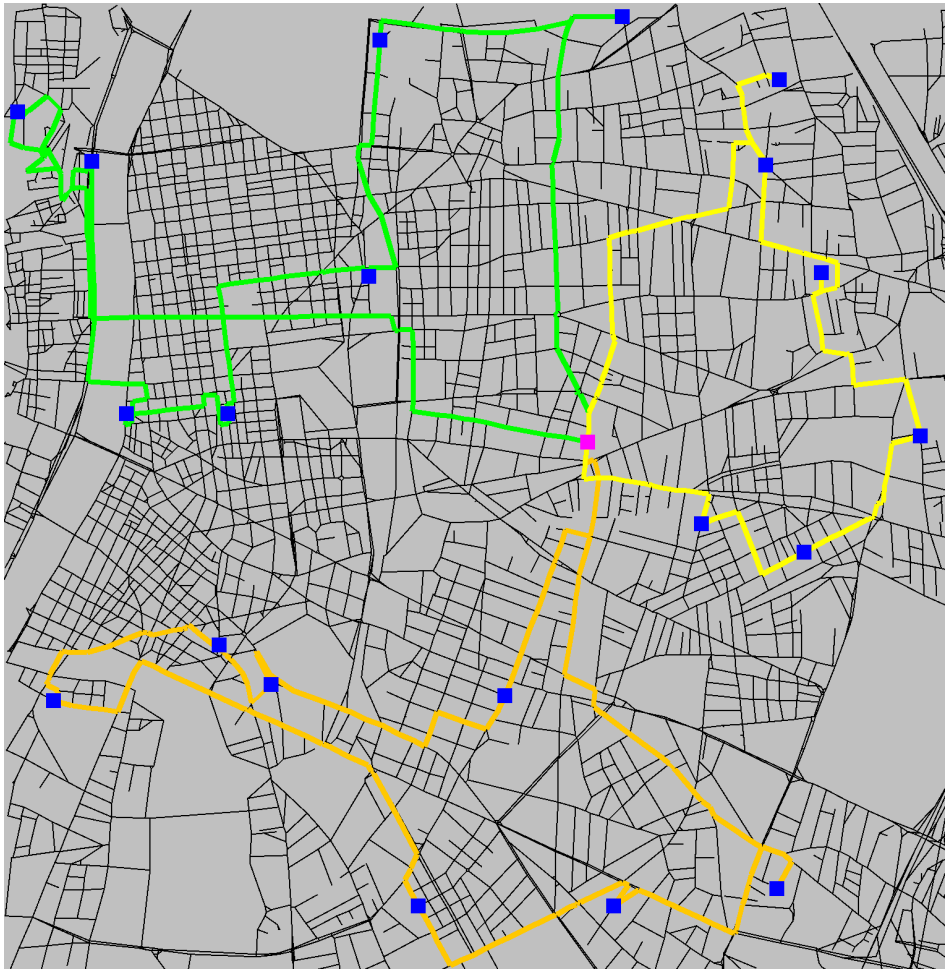
Cas d'utilisation



- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer les tournées pour une demande de livraisons
- Modifier interactivement les tournées

- Chaque point de livraison est visité par exactement un livreur
  $\rightsquigarrow$ Calcul de l'heure de passage du livreur pour prévenir le client
- Tous les livreurs livrent le même nombre de colis (à 1 près)
- La somme des longueurs des tournées est minimale

Cas d'utilisation



- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer les tournées pour une demande de livraisons
- Modifier interactivement les tournées

- Ajouter ou supprimer une livraison
- Changer une livraison de livreur

$\rightsquigarrow$ Mise à jour des horaires de passage

Simplifications par rapport au problème réel

La vitesse est constante

Dans le problème réel, la vitesse dépend du tronçon et de l'heure
Mais quand on circule à vélo, le problème ne se pose pas !

Les seules contraintes sont liées au nombre de livraisons par livreur

Dans le problème réel, il peut y avoir d'autres contraintes :

- Contraintes sur les heures de livraison
- Contraintes de précédence entre livraisons
- ...

La tournée est statique

Dans le problème réel, il faut adapter la tournée pendant la livraison quand les conditions de circulation observées diffèrent des conditions prévues.
Mais quand on circule à vélo, le problème ne se pose pas !

Plan de la partie 5

Présentation du Projet Longue Durée

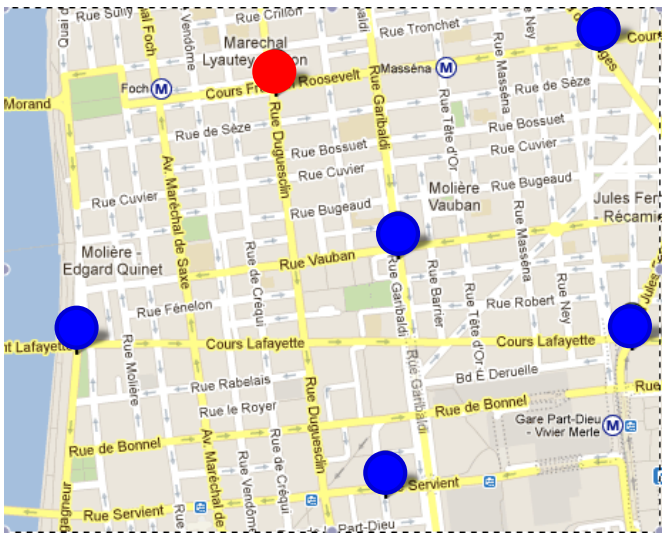
Calcul des tournées en deux étapes

Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de n points de livraison + plan de la ville
- Sortie : G = Graphe complet orienté ayant n sommets

Résolution d'un problème de Vehicle Routing Problem (VRP)

- Entrée : G + nombre de livreurs k
- Sortie : k tours de longueur totale minimale tels que chaque tour passe par n/k livraisons différentes



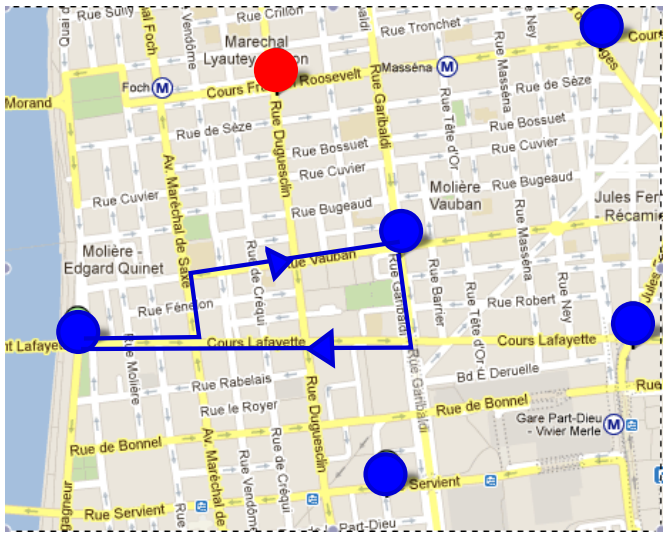
Calcul des tournées en deux étapes

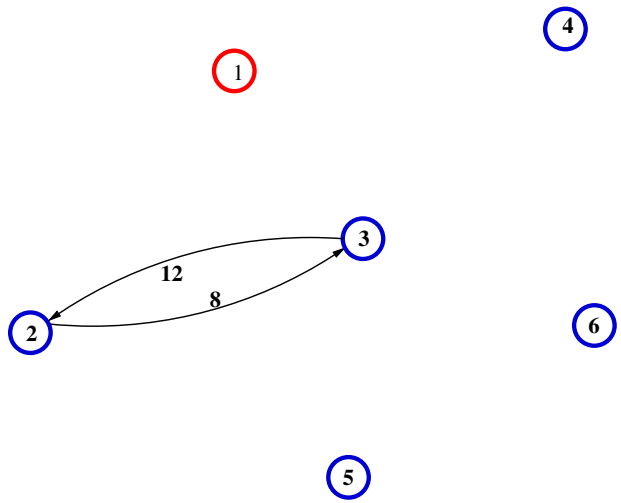
Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de n points de livraison + plan de la ville
- Sortie : G = Graphe complet orienté ayant n sommets

Résolution d'un problème de Vehicle Routing Problem (VRP)

- Entrée : G + nombre de livreurs k
- Sortie : k tours de longueur totale minimale tels que chaque tour passe par n/k livraisons différentes





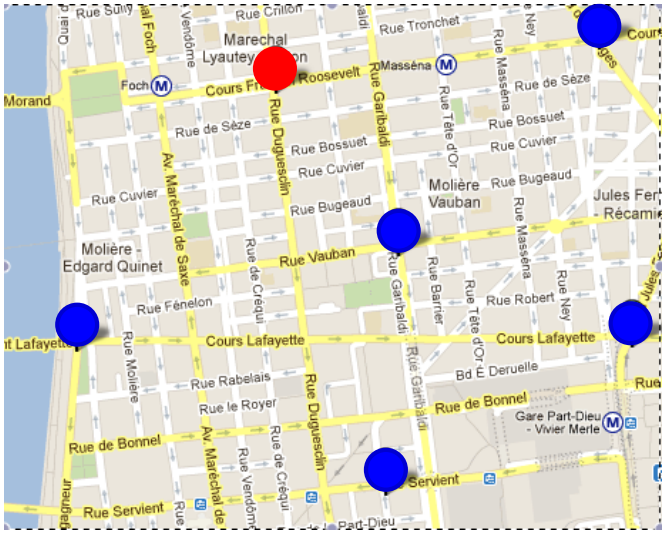
Calcul des tournées en deux étapes

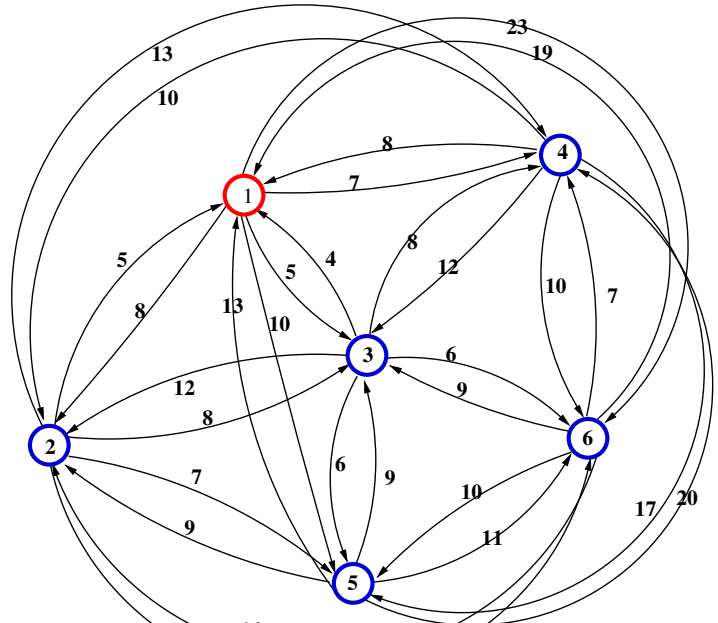
Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de n points de livraison + plan de la ville
- Sortie : G = Graphe complet orienté ayant n sommets

Résolution d'un problème de Vehicle Routing Problem (VRP)

- Entrée : G + nombre de livreurs k
- Sortie : k tours de longueur totale minimale tels que chaque tour passe par n/k livraisons différentes





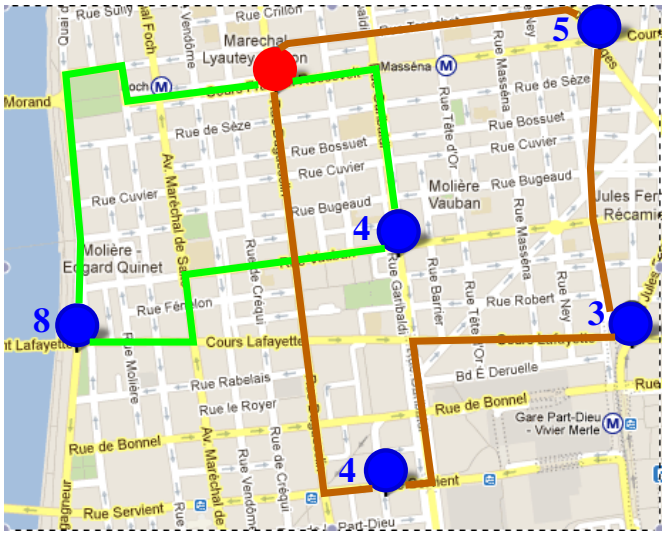
Calcul des tournées en deux étapes

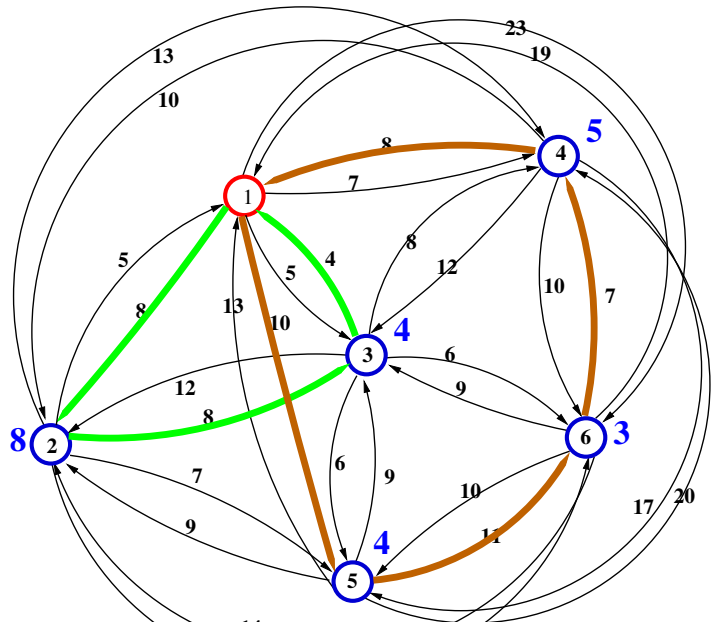
Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de n points de livraison + plan de la ville
- Sortie : G = Graphe complet orienté ayant n sommets

Résolution d'un problème de Vehicle Routing Problem (VRP)

- Entrée : G + nombre de livreurs k
- Sortie : k tours de longueur totale minimale tels que chaque tour passe par n/k livraisons différentes





Comment résoudre un VRP ?

Solution 1 : Résolution en deux étapes

Etape 1 : Choisir les points à livrer pour chaque livreur

Partitionner les points en k sous-ensembles tels que

- Chaque sous-ensemble comporte n/k (ou $1 + n/k$) points
- Les sommets dans un même ss-ens. sont proches les uns des autres

$\rightsquigarrow$ Problème de clustering

Etape 2 : Résoudre un TSP pour chaque livreur

Avantage de l'approche :

Chaque TSP est suffisamment petit pour être facile à résoudre

Inconvénient de l'approche :

La qualité de la solution dépend du clustering
$\rightsquigarrow$ Pas de garantie d'optimalité

Comment résoudre un VRP ?

Solution 2 : Transformer le problème en un TSP

Ajouter $k-1$ sommets fictifs

- Chaque sommet fictif a la même adresse que l'entrepôt
  $\rightsquigarrow$ On a un entrepôt par livreur

Résoudre le TSP

- Passer par un sommet fictif tous les n/k (ou $1+n/k$) sommets
- A chaque fois qu'on passe par un sommet fictif, on change de livreur

Avantage de l'approche :

Garantie d'optimalité

Inconvénient de l'approche :

Un seul TSP comportant $n+k-1$ sommets est bien plus difficile à résoudre que k TSP de n/k sommets

Approches pour la résolution du TSP (rappels 3IF)

Problème NP-difficile

$\rightsquigarrow$ Utiliser des approches appropriées pour explorer l'espace de recherche

Approches complètes (Programmation dynamique, Branch& Bound, ...)

- Exploration exhaustive de l'espace de recherche
  $\rightsquigarrow$ Garantie d'optimalité mais complexité exponentielle
- Mécanismes pour élaguer l'espace de recherche
- Heuristiques pour explorer en premier les zones les plus prometteuses

Approches incomplètes (Recherche locale, Colonies de fourmis, ...)

- Exploration heuristique de l'espace de recherche
  $\rightsquigarrow$ Pas de garantie d'optimalité mais complexité en temps polynomiale
- Mécanismes d'intensification pour guider vers les zones prometteuses
- Mécanismes d'exploration pour guider vers des zones nouvelles

Pour le PLD, vous êtes libres de choisir l'approche que vous voulez

... mais nous vous fournissons une implémentation d'une approche basique

Métaheuristiques pour la résolution du TSP

Recherche locale (Recuit simulé, Recherche taboue, etc) :

- Génération d'une tournée initiale (e.g., avec un algorithme glouton)
- Itérer :
  - Modifications locales (2-opt, par exemple)
  - Mécanismes d'intensification / diversification

Algorithmes génétiques

- Génération d'un ensemble de tournées
- Itérer :
  - Sélectionner les meilleures tournées de l'ensemble
  - Générer de nouvelles tournées à partir des tournées sélectionnées
  - Mise-à-jour de l'ensemble

Approches constructives (ACO, EDA, etc) :

- Construction d'un modèle probabiliste pour générer des tournées
- Itérer :
  - Utilisation du modèle pour générer des tournées
  - Mise-à-jour du modèle en fonction des meilleures tournées

Enumération de toutes les permutations de sommets (rappels 3IF)

```
void enumere(int nbSommets){
    ArrayList<Integer> vus = new ArrayList<Integer>(nbSommets);
    vus.add(0); // le premier sommet visite est 0
    ArrayList<Integer> nonVus = new ArrayList<Integer>();
    for (int i=1; i<nbSommets; i++) nonVus.add(i);
    enumere(0, nonVus, vus);
}

void enumere(int sommetCrt, ArrayList<Integer> nonVus, ArrayList<Integer> vus){
    if (nonVus.size() == 0){
        // vus contient une nouvelle permutation des sommets
    }else{
        for (Integer prochainSommet : nonVus){
            vus.add(prochainSommet);
            nonVus.remove(prochainSommet);
            enumere(prochainSommet, nonVus, vus);
            vus.remove(prochainSommet);
            nonVus.add(prochainSommet);
        }
    }
}
```

Résolution par séparation et évaluation (rappels 3IF)

```
void branchAndBound(int sommetCrt, ArrayList<Integer> nonVus, ArrayList<Integer> vus, int coutVus, int[][] cout){
    if (System.currentTimeMillis() - tpsDebut > tpsLimite) return;
    if (nonVus.size() == 0){ // tous les sommets ont ete visites
        coutVus += cout[sommetCrt][0];
        if (coutVus < coutMeilleureSolution){ // on a trouve une solution meilleure que meilleureSolution
            vus.toArray(meilleureSolution);
            coutMeilleureSolution = coutVus;
        }
    } else if (coutVus+bound(sommetCrt,nonVus,cout) < coutMeilleureSolution){
        Iterator<Integer> it = iterator(sommetCrt, nonVus, cout);
        while (it.hasNext()){
            Integer prochainSommet = it.next();
            vus.add(prochainSommet);
            nonVus.remove(prochainSommet);
            branchAndBound(prochainSommet, nonVus, vus, coutVus+cout[sommetCrt][prochainSommet], cout);
            vus.remove(prochainSommet);
            nonVus.add(prochainSommet);
        }
    }
}
```

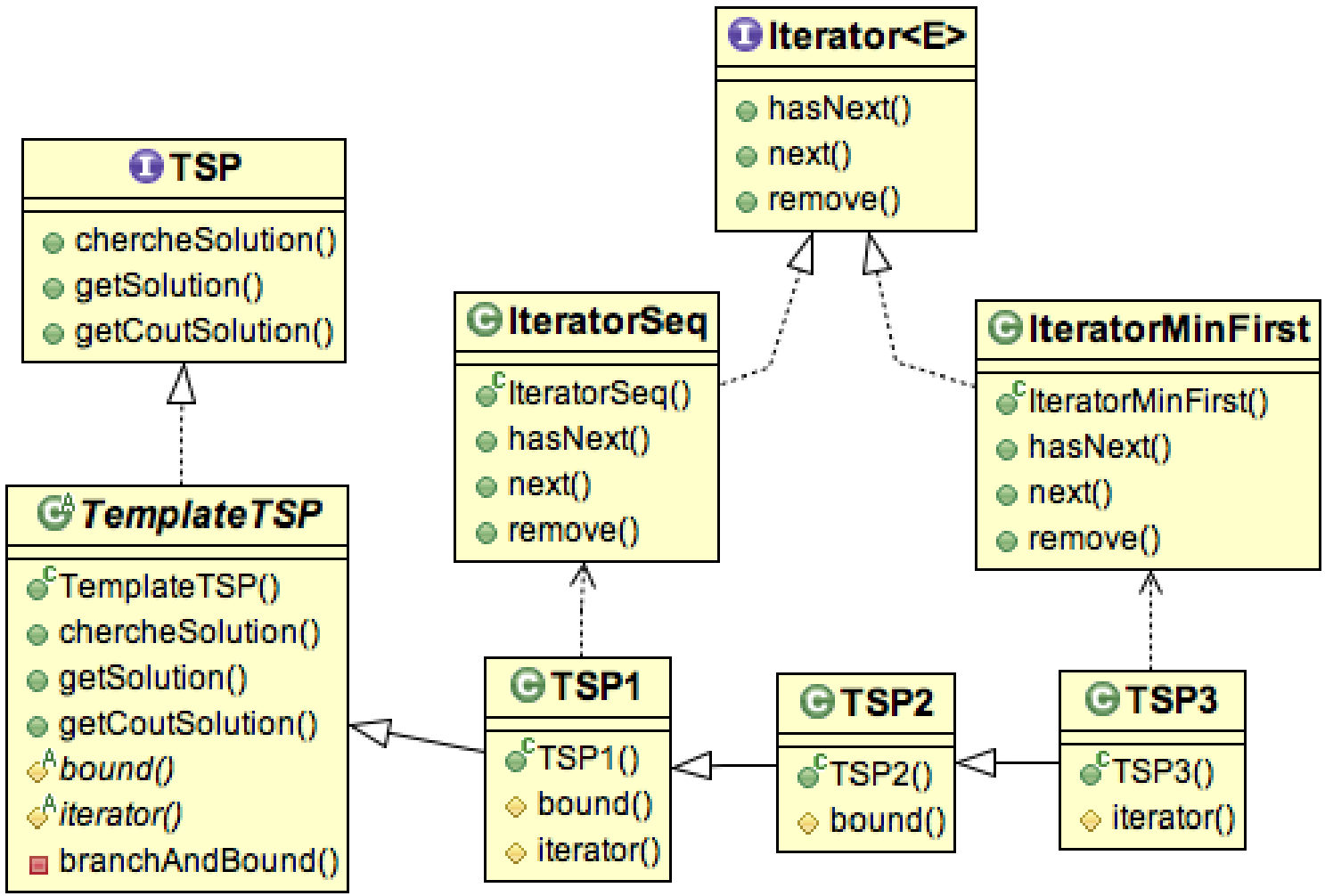
Différentes variantes peuvent être obtenues en changeant :

- L'ordre d'itération sur les sommets de `nonVus` (`iterator`)
- La fonction utilisée pour calculer une borne sur le cout (`bound`)

Comment éviter de dupliquer le code pour chaque variante ?

Pattern GoF Template

- Méthode template (`branchAndBound`) définit l'enchainement des étapes
- Etapes qui varient (`bound, iterator`) = Méthodes abstraites définies dans des sous-classes (`TSP1, TSP2, TSP3`)



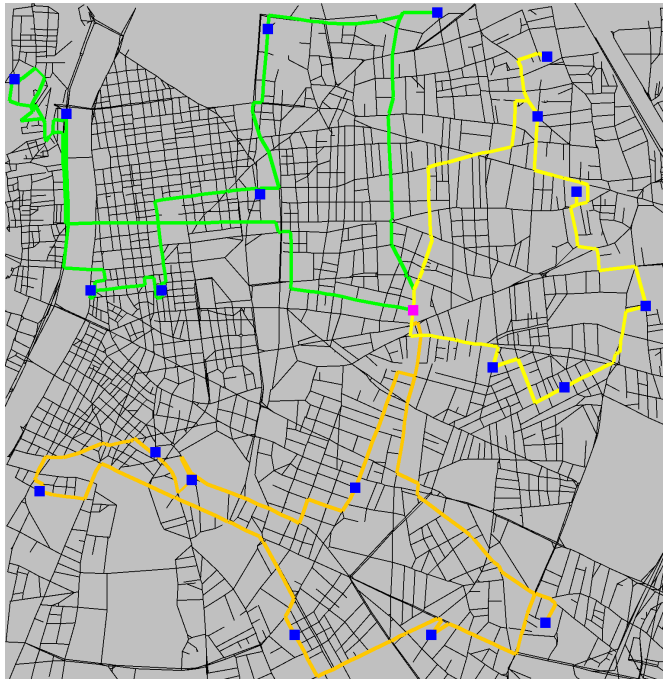
Attention aux symétries !

Solutions symétriques :

Solutions constituées des mêmes tournées, mais dans deux ordres différents

Exemple :

- Solution 1 : vert, puis orange, puis jaune
- Solution 2 : orange, puis vert, puis jaune



Pourquoi "casser" les symétries ?

Pour diviser par $k!$ le nombre de combinaisons à explorer !

Comment "casser" les symétries ?

- Imposer un ordre sur les sommets fictifs : On ne peut visiter le sommet fictif i que si le sommet fictif $i-1$ a été visité
- Imposer un ordre sur le premier sommet suivant chaque sommet fictif : Sommet après sommet fictif $i >$ Sommet après sommet fictif $i-1$

Plan de la partie 5

Présentation du Projet Longue Durée

Organisation du PLD

Travail en hexanome

- Vous êtes libres de choisir votre organisation
  - Chef de projet ? Responsable qualité ?
  - Product owner ? SCRUM manager ?
  - Daily stand-ups ?
  - ...
- Mais vous devrez faire un bilan à la fin !

Mise en œuvre d'un processus de développement itératif (agile)

Itération 1 : Phase d'*inception*

- Durée : 4 séances de 4 heures
- Objectifs :
  - Identifier les principaux cas d'utilisation
  - Analyser les cas d'utilisation les plus prioritaires
  - Concevoir une première architecture
  - Implémenter une première version de votre application

  $\rightsquigarrow$ Présentation au client en fin de quatrième séance

Itérations suivantes : de 1 à 4 itérations (au choix)

A chaque itération :

- Choisir quelques (scénarios de) cas d'utilisation / user story
- Les analyser, les implémenter et les intégrer à votre application

$\rightsquigarrow$ Comparer à chaque fois plannings prévisionnel et effectif

Pour chaque développement de type algorithmique :

$\rightsquigarrow$ Mise en œuvre d'une démarche *Test Driven Development*

Environnement de travail

Ce que nous vous proposons :

- Travail collaboratif et gestion des versions : Git
- Langage : Java $\rightsquigarrow$ JavaDoc + Guide de style préconisé par Oracle (`http://www.oracle.com/technetwork/java/codeconventions-150003.pdf`)
- Bibliothèque pour IHM : Swing (exemple dans PlaCo) ou Java FX
- IDE : Eclipse
- Tests unitaires : JUnit4 (`http://www.junit.org/`)
- Contrôle de la couverture des tests : EclEmma (`http://www.eclemma.org/`)
- Rétro-génération des diagrammes de classes/packages : ObjectAid (`http://www.objectaid.com/`)
- Dessin de diagrammes UML : papier+crayon ou StarUML (`http://staruml.io/`)

Vous pouvez proposer d'autres outils...

...mais vous devez en discuter avec nous !