

# **PLD Agile**

## **Présentation du Projet Longue Durée**

Christine Solnon - Frédérique Laforest

INSA de Lyon - 4IFA

2024/25

# Plan

## Présentation du Projet Longue Durée

- 1 Contexte du PLD : Optimod'Lyon
- 2 Description de l'application
- 3 Algorithmes pour le calcul d'une tournée
- 4 Organisation du PLD

# 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

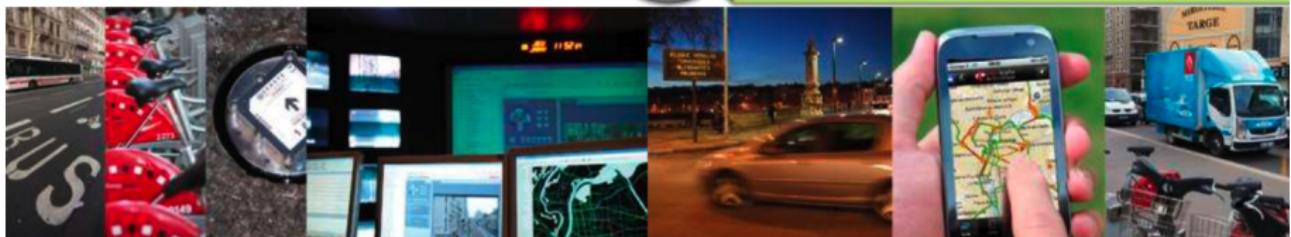
Portail mobilité du Grand Lyon

Entrepôt des données mobilité du territoire

L'optimisation de l'exploitation des réseaux urbains par la prédition 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

## Présentation du Projet Longue Durée

1 Contexte du PLD : Optimod'Lyon

2 Description de l'application

3 Algorithmes pour le calcul d'une tournée

4 Organisation du PLD

# Description de l'application

## Votre mission :

- Concevoir une application pour permettre à des sociétés de livraison de préparer leurs tournées
- Spécificité de cette année : service Pickup & Delivery à vélo !



# Cas d'utilisation



- Charger un plan à partir d'un fichier XML
- Charger un programme de Pickup&Delivery à partir d'un fichier XML
- Calculer la tournée pour un programme de Pickup&Delivery
- Modifier interactivement le programme

Plan = ensemble de tronçons reliant deux intersections

- ~ Chaque tronçon a une longueur et un nom de rue
- ~ Chaque intersection a une latitude et une longitude

# Cas d'utilisation

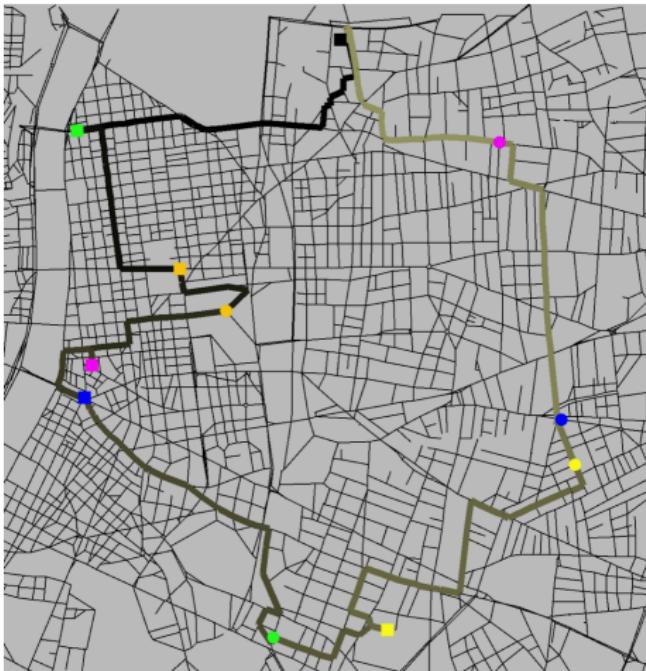


- Charger un plan à partir d'un fichier XML
- Charger un programme de Pickup&Delivery à partir d'un fichier XML
- Calculer la tournée pour un programme de Pickup&Delivery
- Modifier interactivement le programme

Demande de Pickup&delivery =

- Un point d'enlèvement (carré)
- un point de livraison (rond)

# Cas d'utilisation

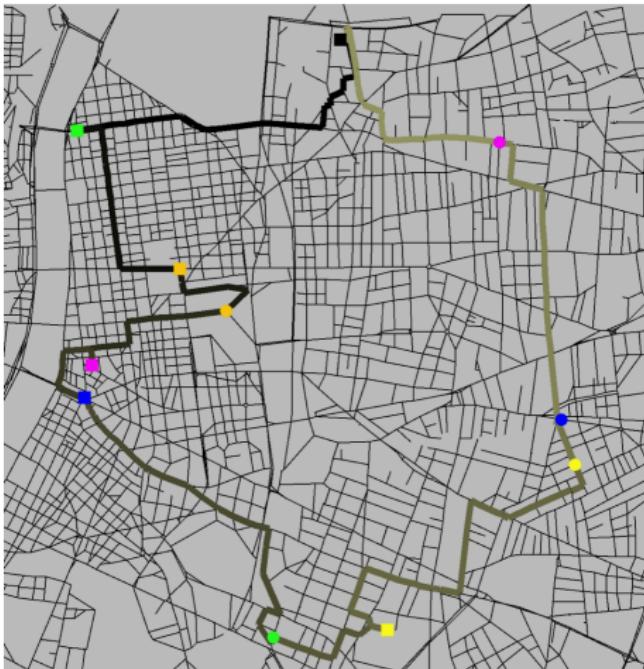


- Charger un plan à partir d'un fichier XML
- Charger un programme de Pickup&Delivery à partir d'un fichier XML
- Calculer la tournée pour un programme de Pickup&Delivery
- Modifier interactivement le programme

Pour chaque up&delivery

- Le point Pickup doit être visité avant le point Delivery  
~ Calcul de l'heure de passage du livreur pour prévenir le client
- La longueur de la tournée doit être minimale

# Cas d'utilisation



- Charger un plan à partir d'un fichier XML
- Charger un programme de Pickup&Delivery à partir d'un fichier XML
- Calculer la tournée pour un programme de Pickup&Delivery
- Modifier interactivement le programme

- Ajouter ou supprimer une livraison
- Déplacer un pickup ou un delivery
- Changer l'ordre de passage aux points

→ 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 des contraintes de précédence

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

- Contraintes sur les heures de livraison
- Contraintes de capacité (poids, nombre de colis...)
- Intégration de plusieurs livreurs (Vehicle Routine Problem)
- ...

## 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 se pose moins !

# Plan

## Présentation du Projet Longue Durée

- 1 Contexte du PLD : Optimod'Lyon
- 2 Description de l'application
- 3 Algorithmes pour le calcul d'une tournée**
- 4 Organisation du PLD

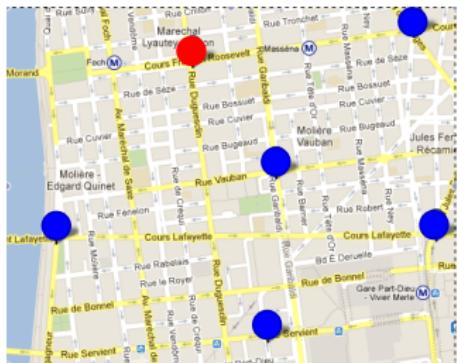
## Calcul d'une tournée en deux étapes

## Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de  $n$  points Pickup et Delivery + plan de la ville
  - Sortie :  $G$  = Graphe complet orienté ayant 1 sommet par point

## Etape 2 : Résoudre un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point
  - Sortie : Ordre de visite des sommets tel que la durée soit minimale  
    ~> Respect des contraintes de précédence entre Pickup et Delivery



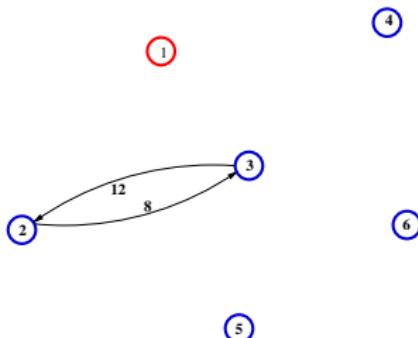
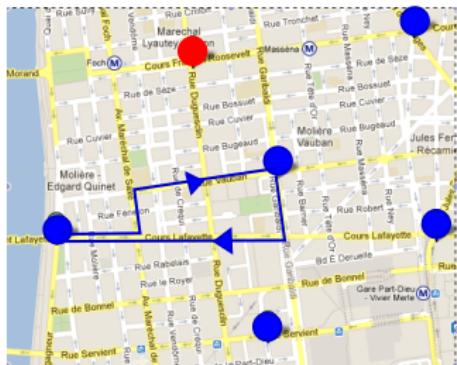
# Calcul d'une tournée en deux étapes

## Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de  $n$  points Pickup et Delivery + plan de la ville
- Sortie :  $G$  = Graphe complet orienté ayant 1 sommet par point

## Etape 2 : Résoudre un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point
- Sortie : Ordre de visite des sommets tel que la durée soit minimale  
~ Respect des contraintes de précédence entre Pickup et Delivery



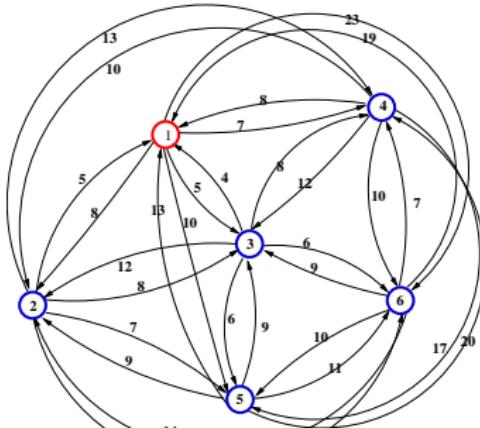
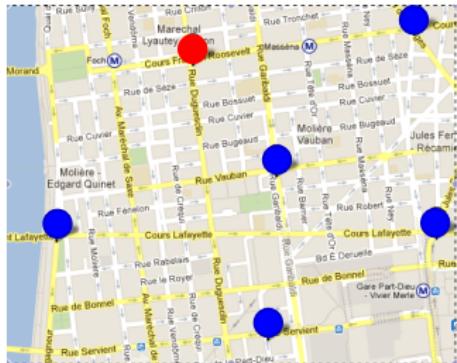
# Calcul d'une tournée en deux étapes

## Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de  $n$  points Pickup et Delivery + plan de la ville
- Sortie :  $G$  = Graphe complet orienté ayant 1 sommet par point

## Etape 2 : Résoudre un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point
- Sortie : Ordre de visite des sommets tel que la durée soit minimale  
~ Respect des contraintes de précédence entre Pickup et Delivery



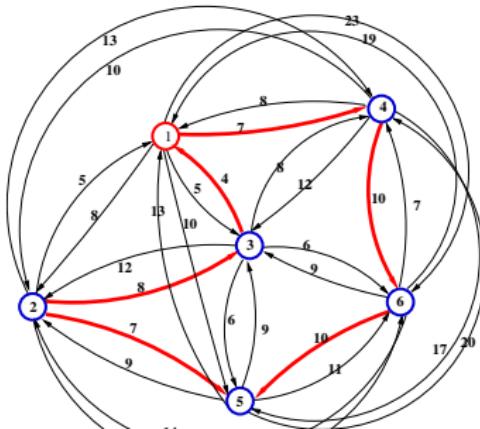
# Calcul d'une tournée en deux étapes

## Etape 1 : Calcul du graphe des plus courts chemins

- Entrée : Ensemble de  $n$  points Pickup et Delivery + plan de la ville
- Sortie :  $G$  = Graphe complet orienté ayant 1 sommet par point

## Etape 2 : Résoudre un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point
- Sortie : Ordre de visite des sommets tel que la durée soit minimale  
~ Respect des contraintes de précédence entre Pickup et Delivery



# Approches pour la résolution du TSP (rappels 3IFA), see [https://en.wikipedia.org/wiki/Travelling\\_salesman\\_problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem)

## Problème NP-difficile

~ Utiliser des approches appropriées pour explorer l'espace de recherche

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

- Exploration exhaustive de l'espace de recherche
  - ~ 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
  - ~ 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

# 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 été visités  
        coutVus += cout[sommetCrt][0];  
        if (coutVus < coutMeilleureSolution){ // on a trouvé 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);  
        }  
    }  
}
```

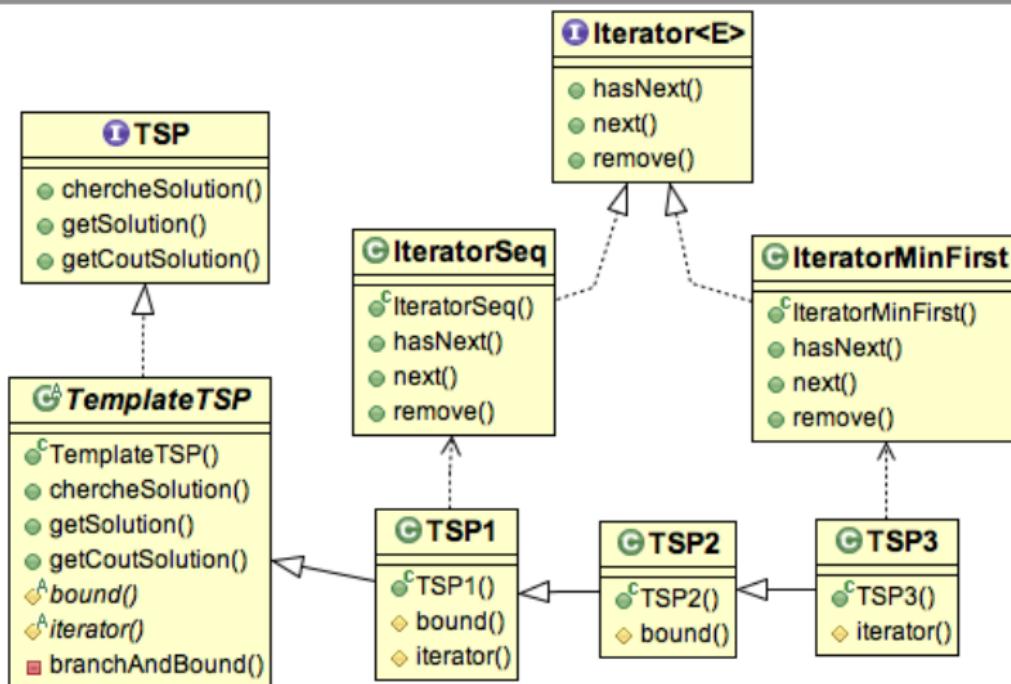
Différentes variantes peuvent être obtenues en changeant :

- 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'enchaînement des étapes
- Etapes qui varient (`bound`, `iterator`) = Méthodes abstraites définies dans des sous-classes (`TSP1`, `TSP2`, `TSP3`)



# Plan

## Présentation du Projet Longue Durée

- 1 Contexte du PLD : Optimod'Lyon
- 2 Description de l'application
- 3 Algorithmes pour le calcul d'une tournée
- 4 Organisation du PLD

# Organisation du PLD

## Travail en \*-nômes

### Méthode Agile

- Itérations
- Transparence, Inspection, Adaptation
- Organisation de l'équipe à définir
  - Durée des itérations, planning global
  - Organisation interne
  - Implication du product owner

### A la fin du projet

- Une livraison formelle du produit avec soutenance
- Un bilan technique, méthodologique et humain

## Itération 1 : Phase de lancement

- Objectifs
  - Identifier les principaux cas d'utilisation
  - Analyser les cas d'utilisation les plus prioritaires
  - Concevoir une première architecture
  - Proposer une interface graphique (maquette "papier")
- ~~> Beaucoup dialoguer avec votre Product Owner
- ~~> Lui faire une présentation en bonne et due forme

## Itérations suivantes

## Itération 1 : Phase de lancement

### Itérations suivantes

A chaque itération

- Choisir un/quelques (scénarios de) cas d'utilisation / user story
  - Les analyser, implémenter, tester, intégrer
  - Faire valider et livrer
- ~ Continuer à dialoguer avec votre Product Owner
- ~ Faire une rétrospective
- ~ Comparer plannings prévisionnel et effectif

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

## Itération 1 : Phase de lancement

## Itérations suivantes

## Pour chaque développement de type algorithmique

~ Mise en œuvre d'une démarche *Test Driven Development*

# Livrailles

## Qu'est-ce qu'on livre ?

- Spécifications = user stories, détaillées pour ce qui est déjà implémenté
- Conception = Architecture logicielle, diagramme de classes & packages commentés (efficacement), choix algorithmiques pour les algorithmes complexes et justifications
- Logiciel = un jar exécutable de la dernière version
- Guide utilisateur
- Suivi projet : burndown/burnup chart comparé, risques rencontrés, rétrospective

## Quand livre-t-on quoi ?

Chaque itération va ajouter/modifier des éléments

~ Tout livrer à chaque itération ~ les livrables itération n+1 sont un update des livrables itération n

# Livraison Finale

## Et pour la livraison finale ?

En plus de ce qu'on livre à chaque fois :

- Une soutenance formelle avec support (slides)
  - présentation du produit et des arbitrages
  - rapport de couverture des tests
  - retour d'expérience (bilan technique, organisationnel et humain)
  - "Et si c'était à refaire ?"
- Le code source avec sa javadoc

# Environnement de travail

## Ce que nous vous proposons :

- Travail collaboratif et gestion des versions : Git
- Langage : Java ~ JavaDoc + Guide de style préconisé par Oracle  
(<http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>)
- Bibliothèque pour IHM : au choix.
- IDE : au choix, tant que toute l'équipe a le même !
- 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 !