

# **PLD Agile**

## **Partie 5 : Présentation du Projet Longue Durée**

Vincent Barrellon  
Elöd Egyed Zsigmond  
Pierre-Edouard Portier  
Christine Solnon

INSA de Lyon - 4IF

2016 / 2017

# Plan de la partie 5

## Présentation du Projet Longue Durée

- 1 Contexte du PLD : Optimod'Lyon
- 2 Description de l'application
- 3 Algorithmes pour le calcul de la 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

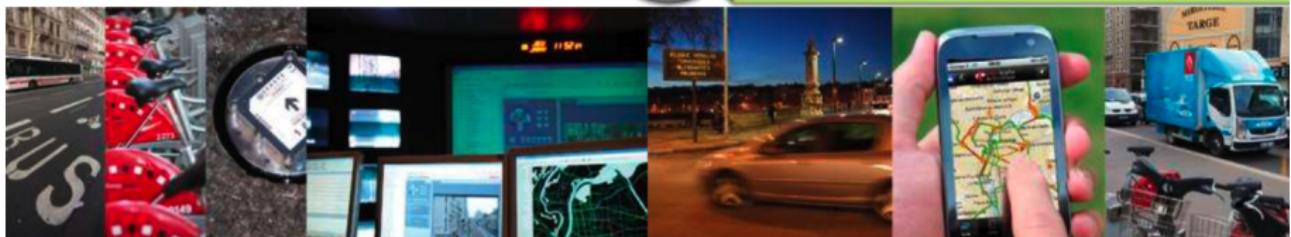
Portail mobilité du Grand Lyon

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 de la partie 5

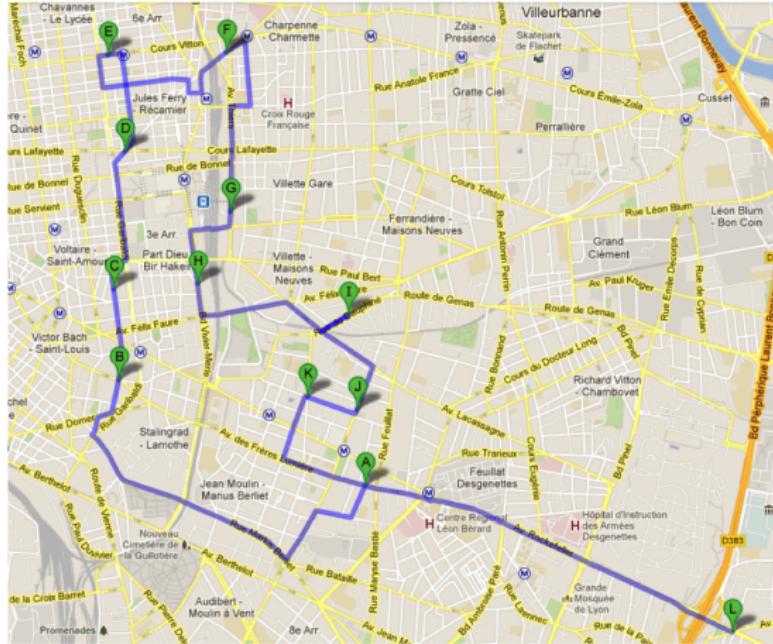
## Présentation du Projet Longue Durée

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

# Description de l'application

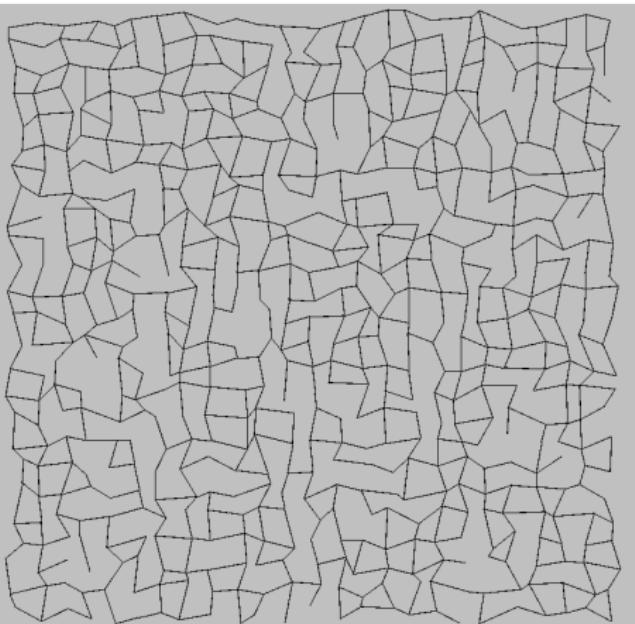
Votre mission : Concevoir une application pour

- Permettre à des sociétés de livraison de préparer leurs tournées
- Guider les chauffeurs de camion pendant leurs livraisons



## Cas d'utilisation :

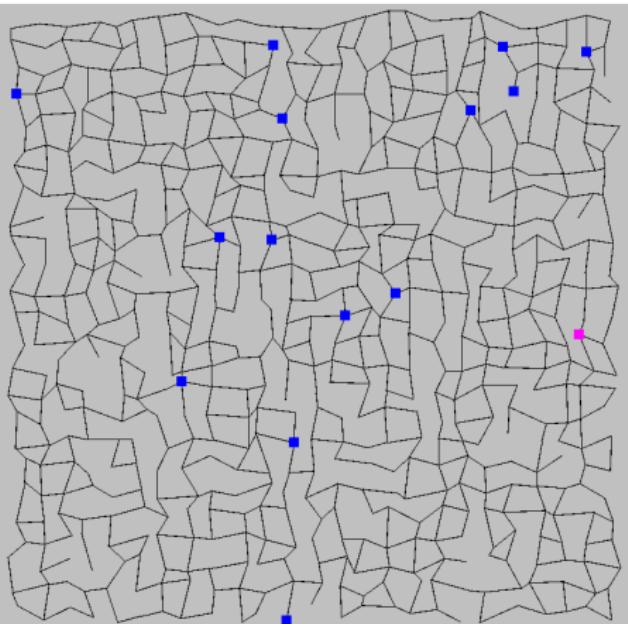
- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer une tournée pour une demande de livraisons
- Modifier interactivement une tournée
- Générer une feuille de route pour le livreur



- Points reliés par des tronçons
- Chaque tronçon a une longueur et une vitesse moyenne

## Cas d'utilisation :

- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer une tournée pour une demande de livraisons
- Modifier interactivement une tournée
- Générer une feuille de route pour le livreur

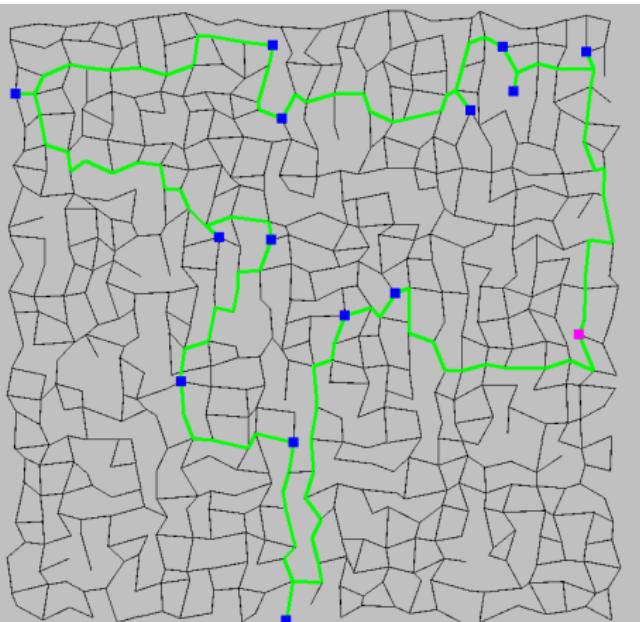


Une demande de livraisons =

- Un entrepôt (carré rose)
- Heure de départ de l'entrepôt donnée
- Des livraisons (carrés bleus)
- Chaque livraison a une durée
- Certaines livraisons ont des plages horaires

## Cas d'utilisation :

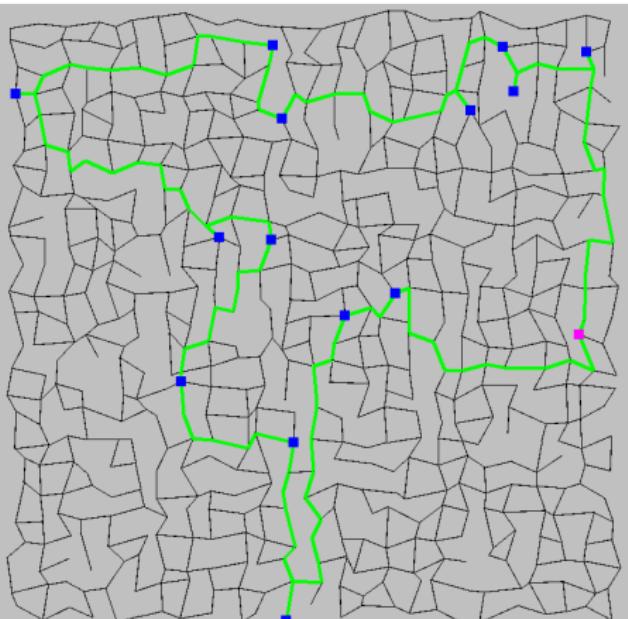
- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer une tournée pour une demande de livraisons
- Modifier interactivement une tournée
- Générer une feuille de route pour le livreur



- Tournée la plus courte partant de (et revenant sur) l'entrepot et passant par chaque livraison
- Livraisons pendant les plages horaires (quand il y en a)
- L'application signale s'il n'est pas possible de respecter toutes les plages horaires

## Cas d'utilisation :

- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer une tournée pour une demande de livraisons
- Modifier interactivement une tournée
- Générer une feuille de route pour le livreur



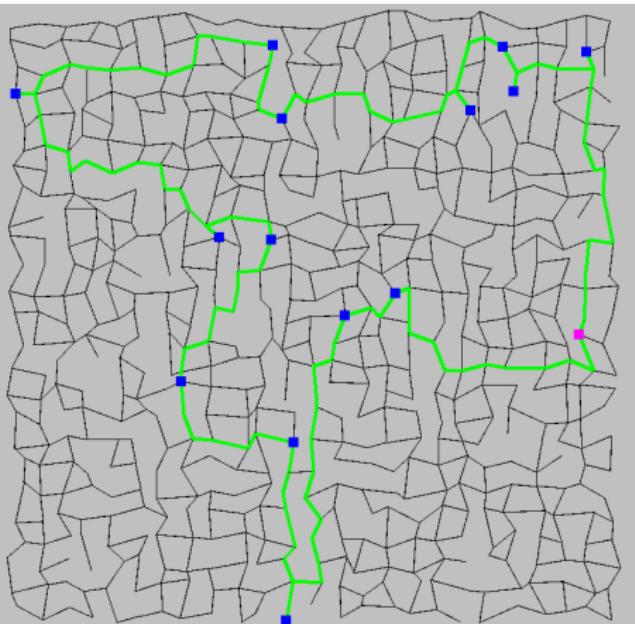
Possibilité de

- Supprimer une livraison
- Ajouter une livraison
- Déplacer une livraison
- Changer la plage horaire d'une livraison

L'application met à jour la tournée (horaires de passage).

## Cas d'utilisation :

- Charger un plan à partir d'un fichier XML
- Charger une demande de livraisons à partir d'un fichier XML
- Calculer une tournée pour une demande de livraisons
- Modifier interactivement une tournée
- Générer une feuille de route pour le livreur



- Description textuelle de la tournée à destination du livreur
- Donne l'itinéraire et les horaires de passage

# Simplifications par rapport au problème réel

## La vitesse sur un tronçon est constante

Dans le problème réel, la vitesse dépend de l'heure de passage

## Les seules contraintes sont liées aux plages horaires

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

## 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.

# Plan de la partie 5

## Présentation du Projet Longue Durée

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

# Calcul de la tournée en deux étapes

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

- Entrée : Ensemble de points de livraison + plan de la ville
- Sortie : Graphe complet orienté ayant 1 sommet par point à livrer

## Résolution d'un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point à livrer
- Sortie : Ordre de visite des sommets tel que la durée soit minimale
  - ~ Respect des plages horaires quand il y en a



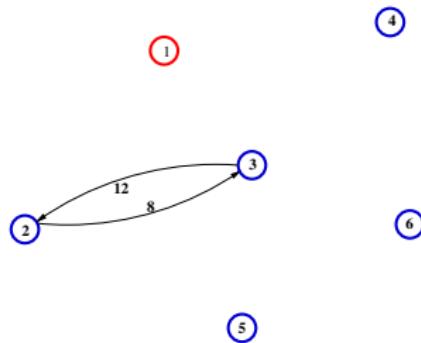
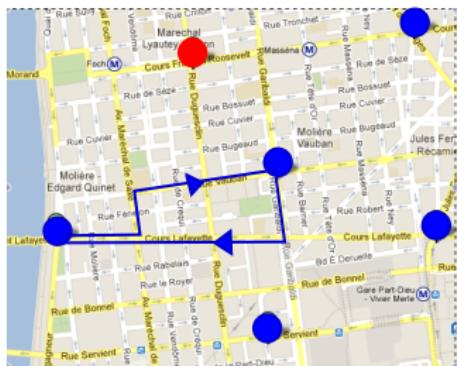
# Calcul de la tournée en deux étapes

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

- Entrée : Ensemble de points de livraison + plan de la ville
- Sortie : Graphe complet orienté ayant 1 sommet par point à livrer

## Résolution d'un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point à livrer
- Sortie : Ordre de visite des sommets tel que la durée soit minimale  
~ Respect des plages horaires quand il y en a



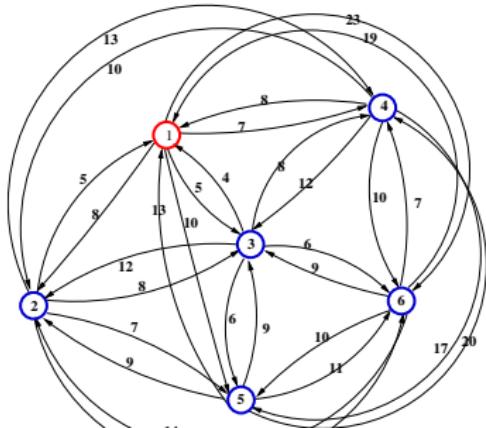
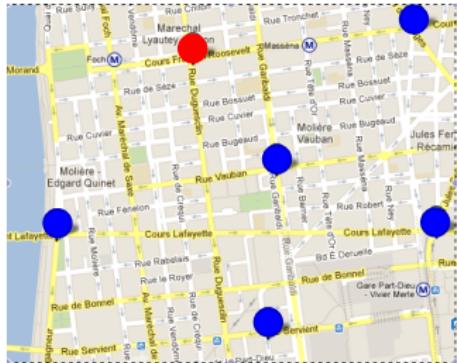
# Calcul de la tournée en deux étapes

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

- Entrée : Ensemble de points de livraison + plan de la ville
- Sortie : Graphe complet orienté ayant 1 sommet par point à livrer

## Résolution d'un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point à livrer
- Sortie : Ordre de visite des sommets tel que la durée soit minimale  
~ Respect des plages horaires quand il y en a



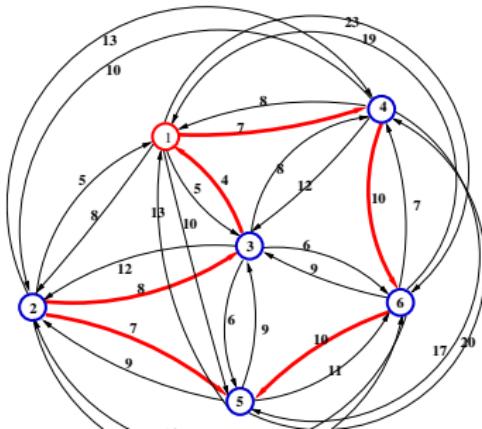
# Calcul de la tournée en deux étapes

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

- Entrée : Ensemble de points de livraison + plan de la ville
- Sortie : Graphe complet orienté ayant 1 sommet par point à livrer

## Résolution d'un problème de voyageur de commerce (TSP)

- Entrée : Graphe complet orienté ayant 1 sommet par point à livrer
- Sortie : Ordre de visite des sommets tel que la durée soit minimale  
~ Respect des plages horaires quand il y en a



# Approches pour la résolution du TSP (rappels 3IF)

## Problème NP-difficile

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

## Approches incomplètes (métaheuristiques)

- 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

## Approches complètes

- 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

**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 (rappels 3IF)

## 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 trouvée 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);  
        }  
    }  
}
```

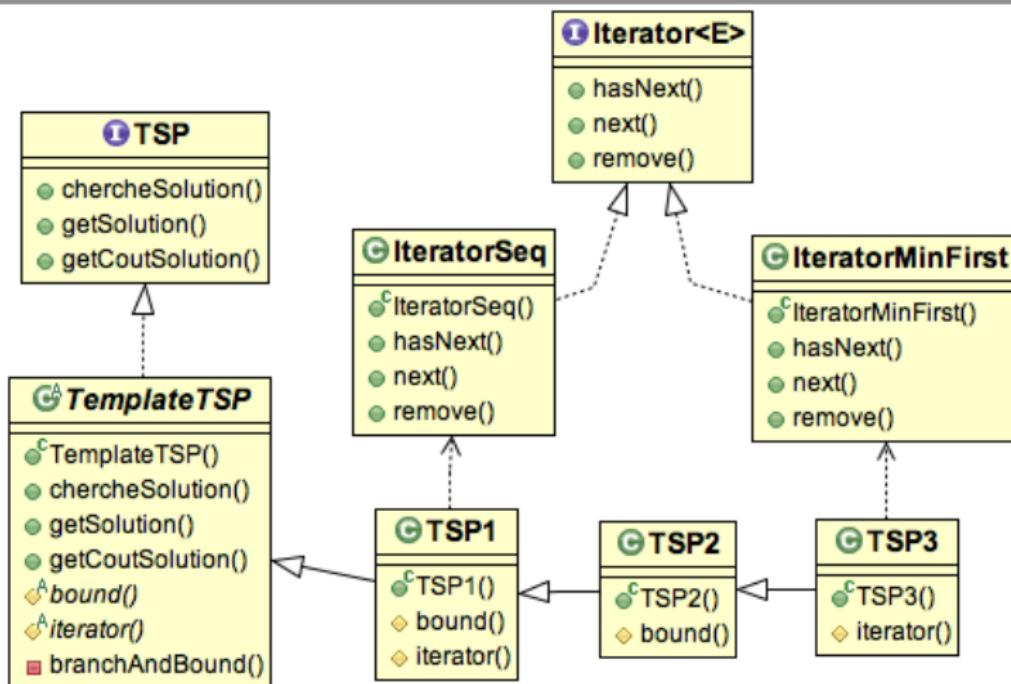
Différentes variantes peuvent être obtenues en changeant :

- 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`)



# Evaluation expérimentale (moy. sur 10 graphes complets)

## Méthodes comparées :

**TSP1** bound = 0, et iterator = séquentiel

**TSP2** bound = somme des couts min, et iterator = séquentiel

**TSP3** bound = somme des couts min, et iterator = cout croissant

nombre de sommets	TSP1	TSP2	TSP3
10	0.06	0.00	0.00
12	0.87	0.00	0.00
14	26.25	0.01	0.00
16	-	0.11	0.04
18	-	0.35	0.12
20	-	1.55	0.46
22	-	13.01	2.47
24	-	-	9.12
26	-	-	17.08
28	-	-	-

## Evaluation expérimentale (moy. sur 10 graphes complets)

### Méthodes comparées :

**TSP1** bound = 0, et iterator = séquentiel

**TSP2** bound = somme des couts min, et iterator = séquentiel

**TSP3** bound = somme des couts min, et iterator = cout croissant

nombre de sommets	TSP1		TSP2		TSP3	
	C	Java	C	Java	C	Java
10	0.00	0.06	0.00	0.00	0.00	0.00
12	0.08	0.87	0.00	0.00	0.00	0.00
14	2.22	26.25	0.00	0.01	0.00	0.00
16	10.49	-	0.01	0.11	0.01	0.04
18	-	-	0.04	0.35	0.02	0.12
20	-	-	0.20	1.55	0.05	0.46
22	-	-	1.28	13.01	0.33	2.47
24	-	-	6.67	-	1.52	9.12
26	-	-	-	-	4.16	17.08
28	-	-	-	-	10.13	-

# Plan de la partie 5

## Présentation du Projet Longue Durée

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

# 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
  - Implémentation d'une première version de votre application
    - ~ Présentation au client en début de cinquième 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
- ~ Comparer à chaque fois plannings prévisionnel et effectif

Pour chaque développement de type algorithme, mise en œuvre d'une démarche couplée *Test Driven Development* et *Design By Contract*

# 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 : Swing (exemple dans PlaCo) ou Java FX
- IDE : Eclipse
- Tests unitaires : JUnit4 (<http://www.junit.org/>)
- *Design By Contract* : CoFoJa
- 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 devrez en discuter avec nous...