# Intelligence Artificielle et Contraintes


UNIVERSITÉ PARIS-SACLAY

FACULTÉ DES SCIENCES - MASTER DATA SCIENCE





## Organisation de visites à domiciles pour un cabinet d'infirmiers

**Professeur: Philippe Chatalic**
**Année Scolaire 2023-2024**


**Binôme: Pablo Mollá Chárlez & Junjie Chen**

# Contents

# 1 Objectif du Projet

Ce rapport présente les modèles et les technologies utilisés dans le cadre du projet d'optimisation des tournées des infirmières libérales en milieu rural. Le but du projet est d'améliorer l'efficience des déplacements des infirmières entre différents patients, en tenant compte de diverses contraintes comme les horaires des soins, les indisponibilités des patients, et la durée des actes médicaux. Nous utilisons des technologies avancées telles que OSRM pour le routage, la Base Adresse Nationale (BAN) pour le géocodage, et le langage de programmation OPL pour l'optimisation des tournées.

## 1.1 Modèles OPL

- **Visits1**:

  - Objectif: Minimiser le temps total de travail pour une infirmière, depuis son départ du cabinet jusqu'à son retour après les dernières visites et la transmission des informations à la Sécurité Sociale.

  - **Hypothèses**: Ce modèle simplifié ignore les contraintes d'espacement spécifiques à chaque visite et les périodes d'indisponibilité des patients, se concentrant uniquement sur l'efficacité des déplacements.

- **Visits2**:

  - Objectif: Tenir compte des contraintes horaires détaillées, y compris les fenêtres de temps pour des soins spécifiques, les exigences d'espacement entre les actes, et les indisponibilités des patients, tout en minimisant le temps total de travail.

  - **Complexité**: Ce modèle est plus complexe que le premier, intégrant la temporalité précise des soins et les indisponibilités pour une gestion plus réaliste et contrainte de la journée de travail.

## 1.2 Technologies Utilisées

### 1.2.1 Open Source Routing Machine (OSRM)

- **Définition**: OSRM est un moteur de routage open source qui utilise les données de OpenStreetMap (OSM) pour calculer des itinéraires optimaux entre des points géographiques. Il est conçu pour des applications de routage à haute performance et permet de calculer des itinéraires, des distances et des temps de trajet rapidement et efficacement.

- **Fonctionnalités Principales**:

  1. **Calcul d'itinéraires**: OSRM fournit le trajet le plus court entre les points de départ et de destination en tenant compte de divers facteurs comme la longueur des routes et les limitations de vitesse.
  2. **Service de table**: Ce service permet de générer une matrice de distances et de temps entre plusieurs points, ce qui est crucial pour les applications nécessitant une analyse complète des trajets, comme l'optimisation des tournées.
  3. **Personnalisation**: Les utilisateurs peuvent personnaliser le comportement du routage pour répondre à des besoins spécifiques, par exemple, en modifiant les poids attribués aux différents types de routes.

- **Utilisation dans le Projet**: Dans notre projet, OSRM est utilisé pour déterminer les meilleurs itinéraires pour les infirmières, en calculant les trajets les plus rapides et les plus courts entre leurs visites à domicile, ce qui est essentiel pour minimiser le temps de déplacement global.

### 1.2.2 Base Adresse Nationale (BAN)

- **Définition**: La Base Adresse Nationale est un projet collaboratif français visant à fournir une base de données unique et ouverte des adresses du territoire français. Elle compile des informations provenant de sources multiples comme l'Institut National de l'Information Géographique et Forestière (IGN), La Poste, et les collectivités territoriales.

- **Fonctionnalités Principales**:

  1. **Géocodage**: La BAN permet de transformer une adresse postale en une paire de coordonnées géographiques (latitude et longitude). Cette fonctionnalité est essentielle pour toutes les applications nécessitant des données spatiales précises.
  2. **API de recherche**: La BAN offre une API qui permet de faire des requêtes pour retrouver des adresses à partir de critères variés, rendant possible l'intégration de ces données dans des applications web ou des systèmes d'information géographique.

- **Utilisation dans le Projet**: La BAN est utilisée pour obtenir les coordonnées géographiques précises des adresses des patients. Ces coordonnées sont ensuite utilisées avec OSRM pour planifier les itinéraires et optimiser les tournées des infirmières.

#### 1.2.3 Optimization Programming Language (OPL)

- **Définition**:

  OPL est un langage de programmation spécifiquement conçu pour la modélisation et la résolution de problèmes d'optimisation. Développé par IBM, il fait partie de la suite logicielle IBM ILOG CPLEX, un des outils les plus puissants pour la résolution de problèmes d'optimisation linéaire et entière mixte.

- **Fonctionnalités Principales**:

  1. **Modélisation intuitive**: OPL permet de décrire des problèmes d'optimisation de manière déclarative, en utilisant une syntaxe proche du langage naturel pour définir les données, les variables, les contraintes, et la fonction objectif.
  2. **Intégration avec les solveurs**: OPL est étroitement intégré avec CPLEX, permettant aux utilisateurs de tirer pleinement parti de ses algorithmes de résolution puissants pour trouver des solutions optimales ou satisfaisantes à des problèmes complexes.

- **Utilisation dans le Projet**: OPL est utilisé pour formaliser et résoudre les différents modèles d'optimisation des tournées infirmières. Les modèles créés avec OPL prennent en compte diverses contraintes opérationnelles et objectifs, tels que la minimisation du temps de déplacement total, la gestion des fenêtres de temps pour les soins, et la répartition équitable du travail entre les infirmières.

Ces technologies sont au cœur du projet, chacune apportant une contribution essentielle à l'efficacité globale du système d'optimisation des tournées pour les infirmières libérales en milieu rural.

### 1.3 Structure du projet

Le projet est structuré autour de différents répertoires qui contiennent les scripts OPL, les données, les librairies nécessaires, et les instances de modèles. Cette organisation facilite le développement, le test, et la maintenance des modèles tout au long du projet.

Chaque partie du projet est essentielle pour atteindre l'objectif d'optimisation des tournées infirmières, utilisant la technologie pour apporter des solutions innovantes aux défis quotidiens dans le secteur des soins à domicile.

# 2 Extraction des données

## 2.1 Fichiers de Texte

### 2.1.1 Structures des données

La présentation des données structurées est cruciale pour expliquer clairement comment le système organise et optimise les tâches quotidiennes. Chaque structure de données représente un aspect spécifique de la gestion des soins infirmiers. Voici comment chaque structure a été définie:

1. **Structure de Journee**: Cette structure définit les horaires de travail de la journée pour les infirmières, permettant de planifier les soins dans les limites horaires disponibles.

   - **Attributs:**
     - infJ: Temps de début de la journée de travail, exprimée en minutes.
     - supJ: Temps de fin de la journée de travail, exprimée en minutes.

2. **Structure de Infirmières**: Représente les infirmières avec leurs plages horaires spécifiques de travail, utilisées pour l'attribution des soins.

   - **Attributs:**
     - nom: Nom de l'infirmière.
     - inf : Début de la disponibilité lors de la journée, exprimée en minutes.
     - sup : Fin de la disponibilité, exprimée en minutes.

3. **Structure de Indisponibilités**: Gère les périodes pendant lesquelles une infirmière ou un patient n'est pas disponible.

   - **Attributs:**
     - nom: Nom de la personne concernée par l'indisponibilité.
     - inf : Début de la période d'indisponibilité, exprimée en minutes.
     - sup : Fin de la période d'indisponibilité, exprimée en minutes.

4. **Structure de Adresses**: Stocke les adresses des patients et du cabinet médical.

   - **Attributs:**
     - patient: Identifie le patient ou utilise "cabinet" pour l'adresse du cabinet médical.
     - adresse : L'adresse physique.

5. **Structure de Actes**: Décrit les différents actes médicaux ou soins que les infirmières peuvent réaliser.

   - **Attributs:**
     - code: Code unique identifiant le type de soin.
     - duree: Durée nécessaire pour effectuer le soin, facilitant ainsi la planification du temps.
     - description: Description détaillée de l'acte.

6. **Structure de Soins/Treatments**: Treatment gère les informations sur les soins administrés à chaque patient avec des créneaux horaires optionnels.

- **Attributs:**
  - name: Nom du patient à soigner.
  - treatment: Code correspondant au soin à appliquer.
  - infJ: Début de la disponibilité du soin, exprimée en minutes.
  - supJ: Fin de la disponibilité du soin, exprimée en minutes.
  - duration: Durée du soins, exprimée en minutes.

7. **Structure de Contraintes**: Contraintes gère les nécessités d'espacement entre différents traitements/soins.

   - **Attributs:**
     - name: Nom du patient.
     - treatment1: Code du premier traitements.
     - duree1: Durée du premier traitement, exprimée en minutes.
     - treatment2: Code du seconde traitement.
     - duree2: Durée du seconde traitement, exprimée en minutes.
     - lowerBound: Début de la contrainte d'espacement entre le premier et deuxième traitement, exprimée en minutes.
     - upperBound: Fin de la contrainte d'espacement entre le premier et deuxième traitement, exprimée en minutes.

8. **Structure de Durations**: Calcule les durées de déplacement entre deux adresses.

   - **Attributs:**
     - From: Adresse de départ
     - To: Adresse d'arrivée.
     - duration: Temps de trajet nécessaire.

### 2.1.2 Lecture et Traitement Initial des Données

1. Première Lecture des Fichiers:

   - Vérification de la présence de 'journee': Le code parcourt chaque fichier pour rechercher des informations sur les horaires de la journée ('journee'). Si cette information est absente, il établit un horaire par défaut de 7h à 21h lors de la seconde lecture.
   - Extraction des Actes : Durant cette première lecture, le code extrait également les informations relatives aux actes médicaux ('acte') et les stocke pour une utilisation ultérieure. Cette étape est cruciale pour préparer les données nécessaires aux traitements sans devoir les relire lors de la seconde passe.

2. Deuxième Lecture et Extraction Complémentaire:

   - Extraction des Adresses ('adresse'): Le script extrait les adresses des patients et les stocke. Cela inclut la manipulation des chaînes de caractères pour s'assurer que les adresses sont correctement formatées et utilisables pour les étapes ultérieures.
   - Extraction des Informations des Infirmières ('infirmiere') : Les disponibilités des infirmières sont lues et traitées. Si seulement le nom est fourni sans horaire, l'horaire par défaut établi est utilisé.
   - Gestion des Indisponibilités ('indisponibilite'): Les périodes pendant lesquelles les infirmières ou les patients ne sont pas disponibles sont enregistrées pour éviter la planification de traitements durant ces moments.
   - Extraction des Soins ('soins'): Cette partie du code traite des informations relatives aux traitements prévus pour les patients, en tenant compte des contraintes d'espacement et des créneaux horaires si spécifiés.

## 2.2 Base Adresse Nationale (BAN)

### 2.2.1 Appels API

Pour interagir avec l'API de la Base d'Adresse Nationale (BAN), nous allons utiliser une série de fonctions spécifiques dans un environnement de JavaScript qui facilitent le traitement des adresses, leur envoi à l'API pour récupération des données géographiques (dont latitude et longitude), et la gestion des réponses obtenues. Voici les principales étapes et fonctions utilisées :

1. Fonction **removeCommas**

   La fonction removeCommas est conçue pour nettoyer les adresses en supprimant les virgules, car elles peuvent interférer avec le format de requête attendu par l'API de la BAN. La suppression des virgules est cruciale pour garantir que les adresses sont interprétées correctement par l'API lors de la construction de la requête.

```
function removeCommas(address) {
    var cleanedAddress = "";
    for (var i = 0; i < address.length; i++) {
        if (address.charAt(i) != ',') {
            // Ajouter le caractère à cleanedAddress
            // s'il n'est pas une virgule
            cleanedAddress += address.charAt(i);
        }
    }
    return cleanedAddress;
}
```

   Cette fonction parcourt chaque caractère de l'adresse, et reconstruit la chaîne sans les virgules. Cela prépare l'adresse pour les étapes de requête suivantes.

2. Fonction **replaceWhitespaceWithPlus**

   La fonction replaceWhitespaceWithPlus prend une adresse où les espaces blancs ont déjà été nettoyés de virgules et les remplace par des signes plus '+'. Ceci est une exigence de formatage pour les URL, car les espaces dans les URL doivent être encodés.

```
function replaceWhitespaceWithPlus(address) {
    var parts = address.split(" ");
    return parts.join("+");
}
```

   Cette fonction utilise la méthode 'split' pour diviser la chaîne originale en un tableau de sous-chaînes en utilisant l'espace comme délimiteur, puis elle utilise 'join' pour reconstruire la chaîne en insérant un signe plus entre chaque élément. Cela rend l'adresse compatible avec les URL.

3. Fonction **extract_BAN**

   La fonction extract_BAN est le coeur du processus d'interaction avec l'API de la BAN. Elle construit la requête, l'envoie à l'API, et gère la réponse.

   Cette fonction prend une adresse formatée, construit l'URL de la requête en ajoutant l'adresse à l'URL de base de l'API, puis utilise 'curl' pour envoyer la requête. La réponse est sauvegardée dans un fichier temporaire que la fonction lit pour convertir la chaîne JSON en un objet utilisable.

   Finalement, le fichier temporaire est supprimé pour nettoyer.

```
function extract_BAN(address) {
    var service = "https://api-adresse.data.gouv.fr/search/?q=";
    var formattedAddress = replaceWhitespaceWithPlus(address);
    var query = service + formattedAddress;

    var answer_tmp_file = "answer_ban.json";
    var fullquery = "curl '" + query + "' >" + answer_tmp_file;
    IloOplExec(fullquery);

    var answerString = file_to_string(answer_tmp_file);
    var answer = parseSimpleJSON(answerString);

    IloOplExec("rm " + answer_tmp_file);

    return answer;
}
```

Après avoir appliqué ces fonctions et obtenu la réponse de l'API sous forme d'objet JSON, l'analyse de cet objet révéle les informations cherchées telles que, les coordonnées géographiques exactes, des détails sur la localité, et potentiellement des erreurs ou des avertissements relatifs aux adresses soumises. Cela permet une utilisation ultérieure pour le calcule des durées des déplacements pour chaque patient et entre le cabinet des infirmières.

### 2.2.2 Analyse des données du BAN

L'idée générale du code est développéé à continuation.

1. **Appel de la Fonction extract_BAN**:
   - Le code initie par annoncer le début de l'appel à l'API BAN.
   - Pour chaque adresse présente dans l'ensemble des adresses (addresses), il nettoie l'adresse en supprimant les virgules grâce à la fonction removeCommas.
   - Il fait ensuite appel à la fonction extract_BAN en passant l'adresse nettoyée pour obtenir les données JSON (jsonBAN) retournées par l'API.
2. **Traitement des Données JSON**:
   - Le code vérifie si les données JSON sont bien reçues et non vides.
   - Si les données sont valides, il parcourt les différents attributs de l'objet JSON pour extraire les informations pertinentes comme la latitude, la longitude, le code postal et la ville.
   - Ces informations sont extraites en naviguant à travers les différentes structures imbriquées (features, geometry, properties) du JSON.
3. **Enregistrement des Données**:
   - Les informations extraites sont ensuite utilisées pour peupler l'ensemble de tuples **addresses_full_info** qui est structuré selon le tuple CadastreAddress.
   - Chaque entrée dans cet ensemble représente une adresse complète avec ses coordonnées géographiques, le lieu, le code postal, la ville et le pays.

## 2.3 Open Source Routing Machine (OSRM)

Pour optimiser les déplacements de l'infirmière dans le cadre de ses visites à domicile, l'utilisation conjointe des fonctions extract_BAN et osrm_table est essentielle. Voici une explication détaillée de la manière dont ces deux fonctions interagissent pour extraire les informations nécessaires et comment elles contribuent à la minimisation du temps total de travail de l'infirmière.

### 2.3.1 Appels API

Une fois les coordonnées obtenues pour chaque adresse visitée par l'infirmière grâce à extract_BAN, elles sont formatées et utilisées comme entrée pour la fonction osrm_table afin de calculer le temps et la distance de déplacement entre ces adresses en utilisant l'API OSRM (Open Source Routing Machine). Cette API permet de calculer des itinéraires optimisés entre plusieurs points géographiques.

1. Fonction **formatCoordinates**

   Cette fonction prend un ensemble de tuples contenant des coordonnées de latitude et longitude, et les formate en une chaîne de caractères où chaque paire de latitude et longitude est séparée par des virgules, et chaque paire est séparée par des points-virgules.

```
function formatCoordinates(coordinatesSet) {
var concatenatedCoordinates = "";
var isFirst = true; // Flag pour aider avec le formatage

for (var i = 0; i < coordinatesSet.length; i++) {
    var coordinate = coordinatesSet[i];
    var formattedCoordinates = coordinates_formatter(
                                coordinate.latitude,
                                coordinate.longitude
                                );
    if (isFirst) {
        concatenatedCoordinates += formattedCoordinates;
        isFirst = false; // Réinitialiser le flag
    } else {
        concatenatedCoordinates += ";" + formattedCoordinates;
    }
return concatenatedCoordinates;
}
```

2. Fonction **coordinates_formatter**

   Cette fonction aide à formater individuellement la latitude et la longitude en une chaîne de caractères, en utilisant simplement une virgule pour les séparer.

```
function coordinates_formatter(latitude, longitude) {
    return latitude.toString() + "," + longitude.toString();
}
```

3. Fonction **osrm_table**

   Après avoir formaté les coordonnées, on utilise ces coordonnées formatées pour effectuer une requête à l'API OSRM. La fonction osrm_table permet de faire cette requête en envoyant les coordonnées au serveur OSRM spécifié et en récupérant les informations de routage.

```
function osrm_table(serie, server) {
    var service = "http://" + server + "/table/v1/driving/";
    var query = service + serie;

    var answer_tmp_file = "answer_osrm.json";
    var fullquery = "curl '" + query + "' >" + answer_tmp_file;
    IloOplExec(fullquery);

    var answerString = file_to_string(answer_tmp_file);
    var answer = parseSimpleJSON(answerString);

    IloOplExec("rm " + answer_tmp_file);

    return answer;
}
```

### 2.3.2 Analyse des données du OSRM

Ce script inclut le processus d'obtention et de traitement des durées de déplacement entre différents lieux en utilisant la fonction osrm_table de l'API OSRM, qui est un service de routage open-source.

1. **Appel de l'API OSRM**

   - Lors du début de l'appel API, le script commence par annoncer l'initiation de l'appel à l'API OSRM, qui est utilisée pour calculer les temps de trajet entre des coordonnées géographiques spécifiques.
   - Pour préparer les coordonnées, le script utilise la fonction **formatCoordinates** pour formater les coordonnées stockées dans l'ensemble 'addresses_full_info'. Cette fonction ajuste le format pour répondre aux exigences de l'API OSRM, qui nécessite les coordonnées en format longitude, latitude.
   - La fonction osrm_table est appelée avec les coordonnées formatées, l'adresse du serveur OSRM et une option spécifiant que l'annotation des durées est requise. Cette fonction renvoie les données JSON (**jsonOSRM_durations**) contenant les informations sur les trajets.

2. **Traitement des Données JSON**

   - En cas d'absence de réponse, le script vérifie si la réponse JSON est valide et si le code de réponse est "Ok". Si ces conditions sont remplies, le traitement des données continue.
   - Pour extraire et traiter les données, le script itère sur les données de destinations et de durées contenues dans le JSON. Pour chaque paire de destinations, il calcule la durée en minutes (arrondie à la minute la plus proche) du point i au point j.
   - Parfois, le nom d'une destination peut être absent dans les données JSON. Le script gère ce cas en attribuant un nom temporaire basé sur les informations dans **addresses_full_info** pour assurer que chaque destination a un identifiant.

3. **Stockage des Informations**

   - Pour chaque paire de destinations valides (i.e., où i est différent de j), le script ajoute la durée calculée dans l'ensemble **allDurations** qui stocke ces informations sous forme de tuples.
   - Chaque entrée dans **allDurations** contient les noms des deux lieux entre lesquels le trajet est mesuré, ainsi que la durée du trajet en minutes. Cette structure permet d'accéder facilement à ces informations pour des analyses ultérieures

# 3 Modelès

## 3.1 Variables partagées

1. patientNames : Elle crée un ensemble de chaînes où chaque chaîne est une valeur unique From extraite de chaque entrée travel dans l'ensemble allDurations. L'ensemble assure que chaque nom est unique, évitant ainsi les doublons.

```
{string} patientNames = {travel.From | travel in allDurations}
```

2. countPatient : Compte le nombre de noms de patients uniques (y compris les lieux spéciaux comme un cabinet) collectés dans l'ensemble patientNames à l'aide de la fonction card().

```
int countPatient = card(patientNames);
```

3. travel_times : Cette ligne initialise une matrice à deux dimensions. La matrice est indexée par les noms des patients à la fois pour les lignes et les colonnes. Pour chaque enregistrement travel dans allDurations, elle associe le patient From à un dictionnaire où le patient To mappe à la duration du déplacement.

```
int travel_times[patientNames][patientNames] = [ travel.From : [travel.To : travel.duration]
                                                 | travel in allDurations];
```

4. rangeMinutes : Définit la plage horaire de travail de l'infirmière basée sur ses heures disponibles les plus tôt (inf) et les plus tard (sup).

```
range rangeMinutes = nurses.inf..nurses.sup;
```

5. duree_transmission : La transmission des données pour chaque patient (sans considérer comme patient le 'cabinet') prend 2 minutes.

```
int duree_transmission = 2*(countPatient -1);
```

### 3.2 Variables de décision

1. **Intervalle de Traitement/Soins**: L'intervalle pour chaque traitement est contraint à commencer au plus tôt à l'heure de début disponible la plus tardive entre l'heure de début de l'infirmière (nurses.inf) et le début souhaité du traitement (pt.infJ).

   De même, l'intervalle doit se terminer au plus tard à l'heure de fin disponible la plus précoce entre l'heure de fin de l'infirmière (nurses.sup) et la fin souhaitée du traitement (pt.supJ).

   La taille de chaque intervalle est définie par la durée du traitement (pt.duration).

```
dvar interval treatmentIntervals[pt in treatments] in
((nurses.inf > pt.infJ)?nurses.inf:pt.infJ) .. ((nurses.sup < pt.supJ)?nurses.sup:pt.supJ)
size pt.duration;
```

2. **Heures de Travail**: Ces variables permettent de modéliser le début et la fin effectifs du travail de l'infirmière sur la journée, fournissant des points de référence pour la planification et l'optimisation des horaires.

```
dvar int workStart in rangeMinutes;
dvar int workEnd in rangeMinutes;
```

3. **Intervalles des contraintes d'espacements**: Chaque intervalle d'espacement est défini pour être entre les heures de début et de fin de l'infirmière.

   La taille de chaque intervalle correspond à la durée du second traitement dans la contrainte, qui fait référence au minimum d'attente entre le premier soin et le deuxième.

```
dvar interval espacementIntervals[espac in contraintes]
              in nurses.inf..nurses.sup
              size espac.duree2;
```

4. **Intervalle de Transmission de données**: L'intervalle est contraint de se situer dans la plage de travail de l'infirmière (rangeMinutes). La taille de cet intervalle est fixée à la durée estimée pour la transmission de données

```
dvar interval transDonnees in rangeMinutes size duree_transmission;
```

5. **Step Function d'Indisponibilité**: Ces transitions marquent les périodes pendant lesquelles les traitements ne doivent pas être planifiés.

```
stepFunction forbiddenTimes[i in indisponibilites] = stepwise{ 1->i.inf; 0->i.sup; 1 };
```

### 3.3 Visits1

Le premier modèle de planification décrit comprend cinq contraintes principales, toutes orientées vers l'objectif de minimiser le temps total de travail de l'infirmière. Voici un aperçu léger des contraintes mentionnées, qui structurent et optimisent la gestion des horaires des traitements infirmiers:

1. **Détermination du début et de la fin du travail**: Cette contrainte fixe le début et la fin de la journée de travail en se basant sur les horaires des traitements planifiés et les temps de déplacement nécessaires pour se rendre au premier patient et revenir du dernier.

2. **Non-Chevauchement des Traitements**: Assure que les traitements programmés pour différents patients ne se superposent pas temporellement, garantissant ainsi que chaque traitement se déroule dans un créneau horaire distinct.

3. **Temps de Déplacement Entre Traitements**: Intègre le temps de déplacement nécessaire entre les traitements d pour s'assurer que l'infirmière dispose de suffisamment de temps pour se déplacer d'un lieu à l'autre sans retard.

4. **Déplacement pour les Transmissions de Données**: Prévoit un intervalle spécifique pour que l' infirmière considère le temps de retourner au cabinet pour effectuer des transmissions de données après le dernier traitement de la journée.

5. **Fin de Traitement Avant Début des Transmissions**: Garantit que tous les traitements se terminent avec suffisamment d'avance pour permettre le début des activités de transmission de données, sans empiéter sur le temps nécessaire pour ces tâches administratives ou informatives.

Ces contraintes structurent le modèle pour qu'il soit à la fois efficace et conforme aux besoins logistiques et opérationnels, optimisant les horaires et réduisant le temps de travail inactif tout en respectant les contraintes logistiques essentielles.

### 3.3.1 Fonction Objective

- **Objectif**: Minimiser la durée totale de la journée de travail.
- **Description**: Cette fonction objective cherche à réduire l'écart entre l'heure de début (workStart) et l'heure de fin (workEnd) du travail. En minimisant cette différence, le modèle vise à optimiser l'efficacité du temps de travail en réduisant les heures improductives et en concentrant les activités dans un intervalle de temps aussi court que possible. À noter que les heures d'attente où l'infirmière ne travaille pas sont considérer dans la durée totale.

```
minimize workEnd - workStart;
```

### 3.3.2 Contraintes du Modèle

1. **Détermination du début et de la fin du travail**:

   - **Description**: Fixer le début du travail au moment où l'infirmière doit partir du "cabinet" pour son premier traitement après avoir pris en compte le temps de trajet, et la fin du travail au moment où elle termine la dernière activité prévue (transDonnees).

```
workStart == min(pt in treatments) (startOf(treatmentIntervals[pt]) - travel_times["cabinet"][pt.name]);
workEnd == endOf(transDonnees);
```

2. **Non-Chevauchement des Traitements**:

   - **Description**: Cette contrainte empêche deux intervalles de traitement de se superposer, garantissant ainsi que chaque traitement est programmé dans un créneau horaire distinct.

```
noOverlap(treatmentIntervals);
```

3. **Temps de Déplacement Entre Traitements**:

   - **Description**: Après chaque traitement, un temps de déplacement doit être inclus avant que l'infirmière puisse commencer le traitement suivant, en tenant compte des distances entre les lieux des patients.

```
forall (i, j in treatments : i != j){
   endOf(treatmentIntervals[i]) + travel_times[i.name][j.name] <= startOf(treatmentIntervals[j])
||  endOf(treatmentIntervals[j]) + travel_times[i.name][j.name] <= startOf(treatmentIntervals[i]);
}
```

4. **Déplacement pour les Transmissions de Données**:

   - **Description**: Cette contrainte garantit qu'il y a suffisamment de temps entre la fin de chaque traitement et le début de la session de transmission de données pour permettre à l'infirmière de voyager de retour au cabinet.

```
forall (i in treatments) {
   endOf(treatmentIntervals[i]) + travel_times[i.name]["cabinet"] <= startOf(transDonnees);
}
```

5. **Fin de Traitement Avant Début des Transmissions**:

   - **Description**: Cette contrainte assure que chaque traitement se termine avant le début des transmissions de données, ajusté par le temps de déplacement.

```
forall (i, j in treatments) {
    endBeforeStart(treatmentIntervals[i], transDonnees, travel_times[i.name][j.name]);
}
```

## 3.4 Visits2

Dans le modèle avancé Visits2 de planification pour les infirmières, deux nouvelles catégories de contraintes ont été ajoutées pour raffiner le système de gestion des horaires.

Ces contraintes sont centrées sur la gestion des indisponibilités (pour les patients et pour l'infirmière) ainsi que sur les contraintes d'espacement entre les traitements. Malgré ces ajouts, l'objectif final demeure toujours le même : minimiser le temps total de travail, c'est-à-dire la différence entre la fin et le début du travail.

### 3.4.1 Contraintes du Modèle

1. **Contraintes d'Espacements**:

   - **Description**: Cette contrainte parcourt toutes les paires de traitements. Lorsqu'une paire correspond aux critères spécifiés dans les contraintes d'espacement (même patient, traitements consécutifs spécifiés), elle impose que l'intervalle entre la fin du premier traitement et le début du second respecte une borne inférieure (lowerBound).

```
forall(e in contraintes) {
   forall(p1, p2 in treatments : p1 != p2) {
       if (p1.name == e.name &&
           p2.name == e.name &&
           p1.treatment == e.treatment1 &&
           p2.treatment == e.treatment2) {
           (endBeforeStart(treatmentIntervals[p1], treatmentIntervals[p2], e.lowerBound));
       }
    }
 }
```

2. **Contraintes d'Indisponibilités - Infirmière**:

   - **Description**: Cette contrainte empêche le début et la fin des traitements durant les périodes où l'infirmière est indisponible. De plus, elle ajuste le début de la journée de travail de l'infirmière pour s'assurer qu'il ne commence pas pendant une période d'indisponibilité, en évaluant si l'heure de début doit être avant ou après ces périodes.

```
forall(pt in treatments) {
   forall(ind in indisponibilites : nurses.name == ind.nom) {
       forbidStart(treatmentIntervals[pt], forbiddenTimes[ind]);
       forbidEnd(treatmentIntervals[pt], forbiddenTimes[ind]);
       workStart >= ind.sup || workStart <= ind.inf;
    }
}
```

3. **Contraintes d'Indisponibilités - Patients**:

   - **Description**: Pour chaque traitement, cette contrainte vérifie s'il existe des périodes d'indisponibilité correspondantes pour le patient concerné. Elle utilise des fonctions échelonnées (forbiddenTimes) qui définissent ces périodes pour interdire le début et la fin des traitements pendant ces intervalles.

```
forall(pt in treatments) {
   forall(ind in indisponibilites : pt.name == ind.nom) {
       forbidStart(treatmentIntervals[pt], forbiddenTimes[ind]);
       forbidEnd(treatmentIntervals[pt], forbiddenTimes[ind]);
    }
}
```

# 4 Analyse des résultats

## 4.1 Modèle Visits1

Le modèle visits1 a été conçu pour établir une base de planification efficace des horaires des infirmières, en se concentrant uniquement sur la minimisation de leur temps de travail quotidien. Ce modèle de base ne considère pas les contraintes d'espacements entre les traitements ni les contraintes d'indisponibilité des patients ou des infirmières.

1. enonce1.dat, enonce3.dat et test_ind_p1.dat : Avec les données de test fournies dans les fichiers enonce1.dat, enonce3.dat et test_ind_p1.dat, le modèle visits1 fonctionne parfaitement bien.

   Ces fichiers contiennent toutes les informations nécessaires pour tester et valider ce modèle initial. Ils incluent des détails sur une infirmière unique, les différents traitements nécessaires pour les patients ainsi que leur durée respective.

   De plus, ils fournissent les adresses correspondantes et les informations concernant le début et la fin de la journée de travail. Les informations sur l'indisponibilité d'un patient spécifique, Eric BRY, sont aussi présentes (que dans enonce1.dat), cependant, elles sont ignorées et considérées les traitements respectives avec un créneau horaire correspondant au temps de travail de l'infirmière.

## 4.2 Modèle Visits2

Le modèle visits2 est une évolution du modèle initial visits1, intégrant des fonctionnalités supplémentaires pour gérer les contraintes d'indisponibilité des patients ainsi que les contraintes d'espacement entre les traitements.

Le modèle visits2 a été testé avec plusieurs fichiers de données pour vérifier sa robustesse et sa capacité à gérer des scénarios complexes :

1. enonce2.dat : Ce fichier a présenté un défi particulier où aucune solution n'était trouvable avec la plage horaire de travail initiale de l'infirmière de 7h à 20h. En ajustant légèrement cette plage à 20h15, le problème devenait faisable, indiquant que les contraintes de temps de travail étaient trop restrictives dans le cas original et que de petits ajustements pouvaient rendre le scénario viable.

2. enonce4.dat : Similaire au test précédent mais avec une contrainte d'indisponibilité ajoutée pour Eric BRY (non disponible de 14h à 16h). Le modèle a également réussi à intégrer cette contrainte sans problème, démontrant sa flexibilité et son efficacité même en présence de restrictions horaires strictes.

3. enonce5_2i.dat & enonce5_3ic.dat : Ces tests introduisent des scénarios avec deux infirmières. Toutefois, notre modèle visits2 n'est pas conçu pour gérer simultanément plusieurs infirmières. Ces cas nécessiteraient le modèle visits3, non codé dans notre situation.

4. test_ind_p1_ip1.dat & test_ind_p2_ii1.dat Similaire au cas précédent, les deux tests présentent des contraintes d'indisponibilité pour rendre la planification plus complexe. Cependant, les résultats obtenus sont cohérents et le modèle par conséquent faisable.

5. medium_10_15_d_1.dat: Dans ce scénario de test, le fichier medium_10_15_d_1.dat comprenait un total de 10 patients et une seule infirmière, nommée Sophie. Le modèle a réussi à trouver une solution cohérente, prouvant sa capacité à organiser efficacement les traitements pour un nombre modéré de patients tout en respectant les plages horaires de travail de l'infirmière et les temps de déplacement nécessaires entre les patients.

6. medium_15_15_d_1.dat & medium_15_20_d_1_i1.dat: Cependant, lors des tests suivants avec les fichiers medium_15_15_d_1.dat et medium_15_20_d_1_i1.dat, le modèle a rencontré des difficultés qui l'ont rendu infaisable. Ces fichiers, représentant des scénarios avec 15 patients, ont exposé des problèmes liés aux temps de déplacement estimés par l'API OSRM.

    - **Temps de Déplacement Excessifs**: La matrice des temps de déplacement obtenue de l'API OSRM montrait des durées de trajet disproportionnées par rapport aux distances réelles, ce qui a été confirmé par des comparaisons avec Google Maps. Ces temps anormalement élevés ont rendu impossible la satisfaction des contraintes de programmation dans le temps alloué, notamment parce que les temps de trajet consommaient une part trop importante des fenêtres de traitement disponibles.
    - **Incohérence des Données de Déplacement**: L'analyse a révélé que les estimations de temps fournies par OSRM pour certains trajets n'étaient pas réalistes, ce qui a affecté négativement la faisabilité du modèle en le rendant trop restrictif pour permettre une programmation efficace.

# 5 Post-Processing Format

## 5.1 Modèle visits1

Pour le modèle visits1 qui gère des visites à domicile sans considérer les contraintes horaires spécifiques, le format de post-processing considéré est le suivant:

```
OBJECTIVE: 769
Début de la journée : 7:35
Fin de la journée : 20:24
Soin pour le patient Jean_DUPONT : MD1 de 9:03 à 9:08 | Original window: [7:00,11:00] | Treatment Duration: 5 minutes
Soin pour le patient Jean_DUPONT : TL1 de 9:08 à 9:23 | Original window: [7:00,11:00] | Treatment Duration: 15 minutes
Soin pour le patient Jean_DUPONT : MD2 de 19:14 à 19:24 | Original window: [19:00,20:30] | Treatment Duration: 10 minutes
Soin pour le patient Jean_DUPONT : TL2 de 19:24 à 19:49 | Original window: [19:00,20:30] | Treatment Duration: 25 minutes
Soin pour le patient Marie_BEL : MD1 de 9:32 à 9:37 | Original window: [8:30,10:00] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : TSA de 9:52 à 9:57 | Original window: [8:30,10:00] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : PS1 de 12:30 à 12:40 | Original window: [12:30,14:00] | Treatment Duration: 10 minutes
Soin pour le patient Marie_BEL : MD1 de 19:00 à 19:05 | Original window: [19:00,20:30] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : IID de 19:58 à 20:03 | Original window: [19:00,20:30] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : CGL de 7:50 à 8:00 | Original window: [7:30,8:00] | Treatment Duration: 10 minutes
Soin pour le patient Eric_BRY : IIM de 8:20 à 8:25 | Original window: [7:00,20:30] | Treatment Duration: 5 minutes
Soin pour le patient Eric_BRY : CSUH de 8:25 à 8:55 | Original window: [7:00,20:30] | Treatment Duration: 30 minutes
Soin pour le patient Eric_BRY : MD1 de 8:15 à 8:20 | Original window: [7:00,20:30] | Treatment Duration: 5 minutes
Transmission des données de 20:18 à 20:24
```

1. **Heures de début et fin de la journée**: Fixe les limites horaires des visites.
2. **Soins par patient**: Identifie le patient et le soin spécifique, avec des codes pour chaque type de soin.
3. **Fenêtre temporelle originale**: Indique les créneaux horaires initialement prévus pour chaque soin.
4. **Durée du traitement**: Temps nécessaire pour chaque soin.
5. **Transmission des données**: Heure de la mise à jour administrative.

### 5.2 Modèle visits2

Le format affiché dans l'image illustre le format de présentation du planning quotidien de soins infirmiers à domicile, adapté au modèle visits2 qui gère des contraintes horaires spécifiques.

Les principaux éléments présentés sont les mêmes qu'avec le modèle visits1 mais avec la présence de fenêtre d'indisponibilité.

```
OBJECTIVE: 758
Début de la journée : 7:35
Fin de la journée : 20:13
Soin pour le patient Jean_DUPONT : TL1 de 8:14 à 8:29 | Original window: [7:00,11:00] | Treatment Duration: 15 minutes
Soin pour le patient Jean_DUPONT : MD1 de 8:09 à 8:14 | Original window: [7:00,11:00] | Treatment Duration: 5 minutes
Soin pour le patient Jean_DUPONT : TL2 de 19:24 à 19:49 | Original window: [19:00,21:00] | Treatment Duration: 25 minutes
Soin pour le patient Jean_DUPONT : MD2 de 19:14 à 19:24 | Original window: [19:00,21:00] | Treatment Duration: 10 minutes
Soin pour le patient Marie_BEL : MD1 de 9:17 à 9:22 | Original window: [8:30,10:00] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : TSA de 8:57 à 9:02 | Original window: [8:30,10:00] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : PS1 de 12:30 à 12:40 | Original window: [12:30,14:00] | Treatment Duration: 10 minutes
Soin pour le patient Marie_BEL : IID de 19:58 à 20:03 | Original window: [19:00,21:00] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : MD1 de 19:00 à 19:05 | Original window: [19:00,21:00] | Treatment Duration: 5 minutes
Soin pour le patient Marie_BEL : CGL de 7:50 à 8:00 | Original window: [7:30,8:00] | Treatment Duration: 10 minutes
Soin pour le patient Eric_BRY : IIM de 8:37 à 8:42 | Original window: [7:00,21:00] | Treatment Duration: 5 minutes
Soin pour le patient Eric_BRY : CSUH de 10:42 à 11:12 | Original window: [7:00,21:00] | Treatment Duration: 30 minutes
Soin pour le patient Eric_BRY : MD1 de 16:00 à 16:05 | Original window: [7:00,21:00] | Treatment Duration: 5 minutes
Unvailability Window for Eric_BRY:  [14:00,16:00]
Transmission des données de 20:07 à 20:13
```

## 6 Conclusion

Les modèles visits1 et visits2 ont été conçus pour optimiser la gestion des horaires des infirmières à domicile, en tenant compte des différentes complexités et contraintes du domaine des soins à domicile. Chaque modèle apporte ses propres forces et limites, adaptées à des scénarios spécifiques.

### 6.1 Modèle visits1

Le modèle visits1 sert de base à la planification efficace des traitements infirmiers, se concentrant sur la minimisation du temps de travail sans intégrer des contraintes complexes telles que les indisponibilités des patients ou les contraintes d'espacement entre les traitements. Ce modèle simplifié a prouvé son efficacité dans des scénarios où les contraintes logistiques sont réduites et où l'objectif principal est de réduire les heures non productives. Le modèle visits1 est idéal pour des contextes où la flexibilité des patients est grande et les traitements relativement simples à planifier.

### 6.2 Modèle visits2

Le modèle visits2, quant à lui, est une évolution de visits1 qui incorpore des fonctionnalités avancées telles que la gestion des indisponibilités des patients et des infirmières ainsi que des contraintes d'espacement entre les traitements. Ce modèle est adapté à des environnements plus complexes où ces facteurs jouent un rôle crucial. Le modèle visits2 a démontré sa capacité à adapter les horaires en fonction de contraintes multiples, offrant une planification plus précise et conforme aux besoins spécifiques des soins à domicile.

Les tests avec différents jeux de données ont révélé que visits2, bien qu'efficace, peut rencontrer des limites lorsque confronté à des estimations irréalistes des temps de déplacement, soulignant l'importance d'utiliser des données précises et vérifiées pour la planification.

### 6.3 Perspectives Futures

Pour les deux modèles, une attention continue à l'exactitude des données d'entrée, en particulier les temps de déplacement, est cruciale. Des améliorations potentielles pourraient inclure l'intégration de techniques plus robustes pour estimer les temps de trajet ou des ajustements dans les algorithmes pour mieux gérer les incertitudes ou les variations dans les données de déplacement.

En résumé, visits1 et visits2 offrent des cadres solides pour la planification des traitements infirmiers à domicile, chacun adapté à différents niveaux de complexité des exigences de soins.