Artificial Intelligence & Constraints

UNIVERSITÉ PARIS-SACLAY

FACULTY OF SCIENCES - MASTER DATA SCIENCE

Scheduling of home visits for a nursing practice

Professeur: Philippe Chatalic

Academic Year 2023-2024

Group: Pablo Mollá Chárlez & Junjie Chen

Contents

1 Objectives of the Project

This report presents the models and technologies used in the project to optimise the rounds of private practice nurses in rural areas. The aim of the project is to improve the efficiency of nurses' journeys between different patients, taking into account various constraints such as care schedules, patient unavailability and the duration of medical procedures. We are using advanced technologies such as OSRM for routing, the National Address Base (BAN) for geocoding, and the OPL programming language to optimise rounds.

1.1 OPL Models

- **Visits1**:

  - Objective: Minimize the total working time for a nurse, from leaving the office to returning after the last visits and transmitting the information to Social Security.

  - **Assumptions**: This simplified model ignores the specific time constraints of each visit and patient availability periods, focusing solely on the efficiency of travel.

- **Visits2**:

  - Objective: Consider detailed time constraints, including time windows for specific care, spacing requirements between procedures, and patient unavailabilities, while minimizing the total working time.

  - **Complexity**: This model is more complex than the first, incorporating the precise timing of care and unavailabilities for a more realistic and constrained management of the workday.

1.2 Used Technologies

1.2.1 Open Source Routing Machine (OSRM)

- **Definition**: OSRM is an open source routing engine that utilizes OpenStreetMap (OSM) data to calculate optimal routes between geographical points. It is designed for high-performance routing applications and allows for the quick and efficient calculation of routes, distances, and travel times.

- **Main Features**:

  1. **Route Calculation**: OSRM provides the shortest route between start and destination points, taking into account various factors such as road lengths and speed limits.
  2. **Table Service**: This service generates a matrix of distances and times between multiple points, crucial for applications that require comprehensive journey analysis, such as tour optimization.
  3. **Customization**: Users can customize the routing behavior to meet specific needs, for example, by altering the weights assigned to different types of roads.

- **Use in the Project**: In our project, OSRM is used to determine the best routes for nurses, calculating the quickest and shortest journeys between their home visits, which is essential for minimizing overall travel time.

1.2.2 Base Adresse Nationale (BAN)

- **Definition**: The Base Adresse Nationale is a French collaborative project aimed at providing a unique and open database of addresses in the French territory. It compiles information from multiple sources such as the National Institute of Geographic and Forest Information (IGN), La Poste, and local authorities.

- **Main Features**:

  1. **Geocoding**: The BAN enables the conversion of postal addresses into geographic coordinates (latitude and longitude). This functionality is essential for all applications requiring precise spatial data.

2. **Search API**: The BAN provides an API that allows queries to find addresses based on various criteria, enabling the integration of these data into web applications or geographic information systems.

- **Use in the Project**: The BAN is used to obtain precise geographic coordinates of patient addresses. These coordinates are then utilized with OSRM to plan routes and optimize the tours of nurses.

1.2.3 Optimization Programming Language (OPL)

- **Definition**:

  OPL is a programming language specifically designed for modeling and solving optimization problems. Developed by IBM, it is part of the IBM ILOG CPLEX software suite, one of the most powerful tools for solving linear and mixed integer optimization problems.

- **Main Features**:

  1. **Intuitive Modeling**: OPL allows for the declarative description of optimization problems, using syntax close to natural language to define data, variables, constraints, and the objective function.
  2. **Integration with Solvers**: OPL is tightly integrated with CPLEX, enabling users to fully leverage its powerful solving algorithms to find optimal or satisfactory solutions to complex problems.

- **Use in the Project**: OPL is used to formalize and solve various optimization models for nurse routing. Models created with OPL consider various operational constraints and objectives, such as minimizing total travel time, managing care time windows, and fairly distributing work among nurses.

These technologies are at the heart of the project, each making an essential contribution to the overall efficiency of the nurse routing optimization system for freelance nurses in rural areas.

1.3 Project Structure

The project is structured around different directories that contain the OPL scripts, data, necessary libraries, and model instances. This organization facilitates the development, testing, and maintenance of models throughout the project.

Each part of the project is essential for achieving the objective of optimizing nurse tours, using technology to bring innovative solutions to daily challenges in the home care sector.

2 Data Extraction

2.1 Text Files

2.1.1 Data Structures

The presentation of structured data is crucial to clearly explain how the system organizes and optimizes daily tasks. Each data structure represents a specific aspect of nursing care management. Here is how each structure has been defined:

1. **Journee Structure**: This structure defines the work schedules for the day for nurses, allowing for planning care within available time limits.

   - **Attributes:**
     - infJ: Start time of the workday, expressed in minutes.
     - supJ: End time of the workday, expressed in minutes.

2. **Nurses Structure**: Represents nurses with their specific work time slots, used for assigning care.

- **Attributes:**
  - nom: Name of the nurse.
  - inf: Start of availability during the day, expressed in minutes.
  - sup: End of availability, expressed in minutes.

3. **Unavailability Structure**: Manages periods during which a nurse or a patient is not available.

   - **Attributes:**
     - nom: Name of the person concerned by the unavailability.
     - inf: Start of the unavailability period, expressed in minutes.
     - sup: End of the unavailability period, expressed in minutes.

4. **Addresses Structure**: Stores the addresses of patients and the medical office.

   - **Attributes:**
     - patient: Identifies the patient or uses "cabinet" for the medical office address.
     - address: The physical address.

5. **Acts Structure**: Describes the various medical acts or care that nurses can perform.

   - **Attributes:**
     - code: Unique code identifying the type of care.
     - duration: Time required to perform the care, thus facilitating time planning.
     - description: Detailed description of the act.

6. **Treatments Structure**: Treatment manages information about care administered to each patient with optional time slots.

   - **Attributes:**
     - name: Name of the patient to be cared for.
     - treatment: Code corresponding to the care to be applied.
     - infJ: Start of the care availability, expressed in minutes.
     - supJ: End of the care availability, expressed in minutes.
     - duration: Duration of the care, expressed in minutes.

7. **Constraints Structure**: Constraints manage the spacing necessities between different treatments/care.

   - **Attributes:**
     - name: Name of the patient.
     - treatment1: Code of the first treatment.
     - duration1: Duration of the first treatment, expressed in minutes.
     - treatment2: Code of the second treatment.
     - duration2: Duration of the second treatment, expressed in minutes.
     - lowerBound: Start of the spacing constraint between the first and second treatment, expressed in minutes.
     - upperBound:
       End of the spacing constraint between the first and second treatment, expressed in minutes.

8. **Durations Structure**: Calculates travel durations between two addresses.

- **Attributes:**
  - From: Departure address.
  - To: Arrival address.
  - duration: Necessary travel time.

2.1.2 Reading and Initial Data Processing

1. First Reading of Files:

   - Checking for 'journee': The code scans each file for information about the day's schedules ('journee'). If this information is missing, it establishes a default schedule from 7 AM to 9 PM during the second reading.
   - Extraction of Acts : During this first reading, the code also extracts information related to medical acts ('act') and stores it for later use. This step is crucial to prepare the necessary data for processing without having to reread it during the second pass.

2. Second Reading and Additional Extraction:

   - Extraction of Addresses ('adresse'): The script extracts patient addresses and stores them. This includes manipulating strings to ensure the addresses are properly formatted and usable for later steps.
   - Extraction of Nurse Information ('infirmiere') : Nurse availabilities are read and processed. If only the name is provided without a schedule, the default schedule established is used.
   - Management of Unavailabilities ('indisponibilite'): The periods during which nurses or patients are unavailable are recorded to avoid scheduling treatments during these times.
   - Extraction of Care ('soins'): This part of the code deals with information related to treatments scheduled for patients, considering spacing constraints and time slots if specified.

2.2 Base Adresse Nationale (BAN)

2.2.1 API Calls

To interact with the API of the Base Adresse Nationale (BAN), we use a series of specific functions in a JavaScript environment that facilitate the processing of addresses, their submission to the API for retrieving geographic data (including latitude and longitude), and managing the responses obtained. Here are the main steps and functions used :

1. Function **removeCommas**

   The function removeCommas is designed to clean addresses by removing commas, as they can interfere with the expected request format by the BAN API. Removing commas is crucial to ensure that addresses are correctly interpreted by the API during the request construction.

```
function removeCommas(address) {
    var cleanedAddress = "";
    for (var i = 0; i < address.length; i++) {
        if (address.charAt(i) != ',') {
            // Add the character to cleanedAddress
            // if it is not a comma
            cleanedAddress += address.charAt(i);
        }
    }
    return cleanedAddress;
}
```

   This function iterates through each character of the address, and rebuilds the string without the commas. This prepares the address for the following request steps.

2. Function **replaceWhitespaceWithPlus**

   The function replaceWhitespaceWithPlus takes an address where white spaces have already been cleaned of commas and replaces them with plus signs '+'. This is a formatting requirement for URLs, as spaces in URLs need to be encoded.

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

   This function uses the 'split' method to divide the original string into an array of sub-strings using space as the delimiter, then uses 'join' to reconstruct the string by inserting a plus sign between each element. This makes the address compatible with URLs.

3. Function **extract_BAN**

   The function extract_BAN is the heart of the process of interacting with the BAN API. It constructs the request, sends it to the API, and handles the response.

   This function takes a formatted address, constructs the request URL by adding the address to the base URL of the API, then uses 'curl' to send the request. The response is saved in a temporary file which the function reads to convert the JSON string into a usable object.

   Finally, the temporary file is removed for cleanup.

```
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;
}
```

After applying these functions and obtaining the API response in the form of a JSON object, analyzing this object reveals the sought information such as the exact geographical coordinates, details about the locality, and potentially errors or warnings related to the submitted addresses. This allows for subsequent use in calculating the travel durations for each patient and between the nurses' office.

2.2.2 Analysis of BAN Data

The general idea of the code is developed as follows.

1. **Call of the extract_BAN Function**:
   - The code begins by announcing the start of the call to the BAN API.
   - For each address present in the address set (addresses), it cleans the address by removing commas using the removeCommas function.
   - It then calls the extract_BAN function by passing the cleaned address to obtain the JSON data (jsonBAN) returned by the API.
2. **JSON Data Processing**:

- The code checks if the JSON data is well received and not empty.
- If the data is valid, it traverses the different attributes of the JSON object to extract relevant information such as latitude, longitude, postal code, and city.
- These details are extracted by navigating through the various nested structures (features, geometry, properties) of the JSON.

3. **Data Recording**:

   - The extracted information is then used to populate the set of tuples **addresses_full_info** which is structured according to the CadastreAddress tuple.
   - Each entry in this set represents a complete address with its geographical coordinates, place, postal code, city, and country.

2.3 Open Source Routing Machine (OSRM)

To optimize the nurse's travels during her home visits, the joint use of the functions extract_BAN and osrm_table is essential. Here is a detailed explanation of how these two functions interact to extract the necessary information and how they contribute to minimizing the total work time of the nurse.

2.3.1 API Calls

Once the coordinates are obtained for each address visited by the nurse thanks to extract_BAN, they are formatted and used as input for the function osrm_table to calculate the travel time and distance between these addresses using the OSRM API (Open Source Routing Machine). This API allows for calculating optimized routes between multiple geographical points.

1. Function **formatCoordinates**

   This function takes a set of tuples containing latitude and longitude coordinates, and formats them into a string where each pair of latitude

   and longitude is separated by commas, and each pair is separated by semicolons.

```
function formatCoordinates(coordinatesSet) {
var concatenatedCoordinates = "";
var isFirst = true; // Flag to help with formatting

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; // Reset the flag
    } else {
        concatenatedCoordinates += ";" + formattedCoordinates;
    }
return concatenatedCoordinates;
}
```

2. Function **coordinates_formatter**

   This function helps to format latitude and longitude individually into a string, simply using a comma to separate them.

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

3. Function **osrm_table**

   After the coordinates are formatted, these formatted coordinates are used to make a request to the OSRM API. The function osrm_table allows this request by sending the coordinates to the specified OSRM server and retrieving routing information.

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

    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 Analysis of OSRM Data

This script includes the process of obtaining and processing travel durations between different locations using the osrm_table function of the OSRM API, which is an open-source routing service.

1. **OSRM API Call**

   - At the start of the API call, the script begins by announcing the initiation of the call to the OSRM API, which is used to calculate travel times between specific geographic coordinates.
   - To prepare the coordinates, the script uses the **formatCoordinates** function to format the coordinates stored in the 'addresses_full_info' set. This function adjusts the format to meet the requirements of the OSRM API, which requires coordinates in longitude, latitude format.
   - The osrm_table function is called with the formatted coordinates, the OSRM server address, and an option specifying that the annotation of durations is required. This function returns JSON data (**jsonOSRM_durations**) containing information about the journeys.

2. **JSON Data Processing**

   - In the absence of a response, the script checks if the JSON response is valid and if the response code is "Ok". If these conditions are met, the data processing continues.
   - To extract and process the data, the script iterates over the destination and duration data contained in the JSON. For each pair of destinations, it calculates the duration in minutes (rounded to the nearest minute) from point i to point j.
   - Sometimes, the name of a destination may be absent in the JSON data. The script handles this case by assigning a temporary name based on the information in **addresses_full_info** to ensure that each destination has an identifier.

3. **Storing Information**

   - For each pair of valid destinations (i.e., where i is different from j), the script adds the calculated duration to the **allDurations** set, which stores these details in tuple form.
   - Each entry in **allDurations** contains the names of the two locations between which the journey is measured, as well as the duration of the journey in minutes. This structure allows easy access to this information for further analysis.

3 Models

3.1 Shared Variables

1. patientNames: It creates a set of strings where each string is a unique From value extracted from each travel entry in the allDurations set. The set ensures that each name is unique, thus avoiding duplicates.

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

2. countPatient: Counts the number of unique patient names (including special places like a medical office) collected in the patientNames set using the card() function.

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

3. travel_times: This line initializes a two-dimensional matrix. The matrix is indexed by patient names for both rows and columns. For each travel record in allDurations, it associates the patient From to a dictionary where the patient To maps to the duration of travel.

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

4. rangeMinutes: Defines the nurse's work time range based on her earliest (inf) and latest (sup) available time (in minutes).

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

5. duree_transmission: The data transmission for each patient (not considering the 'office' as a patient) takes 2 minutes.

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

3.2 Decision Variables

1. **Treatment/Care Interval**: The interval for each treatment is constrained to start no earlier than the latest available start time between the nurse's start time (nurses.inf) and the desired start time of the treatment (pt.infJ).

   Similarly, the interval must end no later than the earliest available end time between the nurse's end time (nurses.sup) and the desired end time of the treatment (pt.supJ).

   The duration of each interval is defined by the duration of the treatment (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. **Working Hours**: These variables model the actual start and end times of the nurse's workday, providing reference points for scheduling and optimization.

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

3. **Spacing Constraint Intervals**: Each spacing interval is defined to be between the nurse's start and end times. The duration of each interval corresponds to the duration of the second treatment in the constraint, which refers to the minimum wait between the first and second care.

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

4. **Data Transmission Interval**: The interval is constrained to be within the nurse's working range (rangeMinutes). The duration of this interval is set to the estimated time for data transmission.

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

5. **Step Function for Unavailability**: These transitions mark the periods during which treatments should not be scheduled.

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

3.3 Visits1

The first scheduling model described includes five main constraints, all aimed at minimizing the total working time of the nurse. Here is an overview of the mentioned constraints, which structure and optimize the management of nursing treatment schedules:

1. **Determination of Work Start and End**: This constraint fixes the start and end of the workday based on the schedules of planned treatments and the necessary travel time to reach the first patient and return from the last.
2. **Non-Overlap of Treatments**: Ensures that scheduled treatments for different patients don't temporally overlap, thus guaranteeing that each treatment occurs in a distinct time slot.
3. **Travel Time Between Treatments**: Incorporates the necessary travel time between treatments for different patients to ensure that the nurse has enough time to move from one location to another without delay.
4. **Travel for Data Transmission**: Provides for a specific interval for the nurse to account for the time to return to the office to perform data transmissions after the last treatment of the day.
5. **End of Treatment Before Start of Transmissions**: Ensures that all treatments end with sufficient advance to allow the start of data transmission activities, without encroaching on the time needed for these administrative or informative tasks.

These constraints structure the model to be both efficient and in line with logistical and operational needs, optimizing schedules and reducing idle working hours while respecting essential logistical constraints.

3.3.1 Objective Function

- **Objective**: Minimize the total duration of the workday.
- **Description**: This objective function seeks to reduce the gap between the start time (workStart) and end time (workEnd) of work. By minimizing this difference, the model aims to optimize the efficiency of working hours by reducing unproductive hours and concentrating activities within as short a time interval as possible. Note that waiting hours where the nurse is not working are considered in the total duration.

```
minimize workEnd - workStart;
```

3.3.2 Model Constraints

1. **Determination of Work Start and End**:
   - **Description**: Fix the start of work at the moment the nurse needs to leave the "office" for her first treatment after accounting for travel time, and the end of work at the moment she finishes the last planned activity (transDonnees).

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

2. **Non-Overlap of Treatments**:
   - **Description**: This constraint prevents two treatment intervals from overlapping, ensuring that each treatment is scheduled in a distinct time slot.

```
noOverlap(treatmentIntervals);
```

3. **Travel Time Between Treatments**:

   - **Description**: After each treatment, a travel time must be included before the nurse can start the next treatment, taking into account the distances between patient locations.

```
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. **Travel for Data Transmissions**:

   - **Description**: This constraint ensures there is sufficient time between the end of each treatment and the start of the data transmission session to allow the nurse to travel back to the office.

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

5. **End of Treatment Before Start of Transmissions**:

   - **Description**: This constraint ensures that each treatment ends before the start of data transmissions, adjusted by travel time, although its exact formulation may need clarification to be sure of its precise intention.

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

3.4 Visits2

In the advanced Visits2 scheduling model for nurses, two new categories of constraints have been added to refine the schedule management system.

These constraints focus on managing unavailabilities (for both patients and the nurse) as well as the spacing constraints between treatments. Despite these additions, the ultimate goal remains the same: to minimize the total working time, that is, the difference between the end and the start of work.

3.4.1 Model Constraints

1. **Spacing Constraints**:

   - **Description**: This constraint iterates over all pairs of treatments. When a pair meets the criteria specified in the spacing constraints (same patient, specified consecutive treatments), it imposes that the interval between the end of the first treatment and the start of the second respects a lower bound (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. **Nurse Unavailability Constraints**:

- **Description**: This constraint prevents the start and end of treatments during periods when the nurse is unavailable. Additionally, it adjusts the start of the nurse's workday to ensure it does not begin during an unavailability period, by assessing whether the start time should be before or after these periods.

```
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. **Patient Unavailability Constraints**:

- **Description**: For each treatment, this constraint checks for any corresponding unavailability periods for the concerned patient. It uses step functions (forbiddenTimes) that define these periods to prohibit the start and end of treatments during these intervals.

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

4 Analysis of Results

4.1 Model Visits1

The Visits1 model was designed to establish a base for efficient scheduling of nurse shifts, focusing solely on minimizing their daily working time. This basic model does not consider spacing constraints between treatments or unavailability constraints for patients or nurses.

1. enonce1.dat, enonce3.dat, and test_ind_p1.dat: With the test data provided in the files enonce1.dat, enonce3.dat, and test_ind_p1.dat, the visits1 model performs exceptionally well. These files contain all the necessary information to test and validate this initial model. They include details about a single nurse, the different treatments required for patients along with their respective durations, and also provide the corresponding addresses and information about the start and end of the workday. Information about a specific patient's unavailability, Eric BRY, is also present (only in enonce1.dat), however, it is ignored and the respective treatments are considered with a time slot corresponding to the nurse's working hours.

4.2 Model Visits2

The Visits2 model is an evolution of the initial Visits1 model, incorporating additional features to manage patient unavailability constraints as well as spacing constraints between treatments.

The Visits2 model was tested with several data files to verify its robustness and capability to handle complex scenarios:

1. enonce2.dat: This file presented a particular challenge where no solution was found with the initial work hour range of the nurse from 7 AM to 8 PM. By slightly adjusting this range to 8:15 PM, the scenario became feasible, indicating that the work time constraints were too restrictive in the original case and that minor adjustments could make the scenario viable.

2. enonce4.dat: Similar to the previous test but with an added unavailability constraint for Eric BRY (unavailable from 2 PM to 4 PM). The model also successfully integrated this constraint without issue, demonstrating its flexibility and effectiveness even in the presence of strict time restrictions.

3. enonce5_2i.dat & enonce5_3ic.dat: These tests introduce scenarios with two nurses. However, our Visits2 model is not designed to simultaneously handle multiple nurses. These cases would require the Visits3 model, which is not coded in our current setup.

4. test_ind_p1_ip1.dat & test_ind_p2_ii1.dat Similar to the previous case, both tests introduce unavailability constraints to make scheduling more complex. However, the results obtained are consistent, and the model is consequently feasible.

5. medium_10_15_d_1.dat: In this test scenario, the file medium_10_15_d_1.dat included a total of 10 patients and a single nurse named Sophie. The model successfully found a consistent solution, proving its ability to efficiently organize treatments for a moderate number of patients while respecting the nurse's work hours and the necessary travel times between patients.

6. medium_15_15_d_1.dat & medium_15_20_d_1_i1.dat: However, in the following tests with the files medium_15_15_d_1 and medium_15_20_d_1_i1.dat, the model encountered difficulties that rendered it infeasible. These files, representing scenarios with 15 patients, exposed problems related to the travel times estimated by the OSRM API.

   - **Excessive Travel Times**: The travel time matrix obtained from the OSRM API showed travel durations disproportionate to the actual distances, which was confirmed by comparisons with Google Maps. These abnormally high times made it impossible to meet the scheduling constraints within the allotted time, particularly because the travel times consumed too large a portion of the available treatment windows.
   - **Inconsistency in Travel Data**: The analysis revealed that the time estimates provided by OSRM for certain journeys were not realistic, negatively affecting the feasibility of the model by making it too restrictive to allow effective scheduling.

5 Post-Processing Format

5.1 Model Visits1

For the Visits1 model, which handles home visits without considering specific time constraints, the post-processing format considered is as follows:

```
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. **Start and End Times of the Day**: Sets the time boundaries for the visits.
2. **Care per Patient**: Identifies the patient and the specific care, with codes for each type of care.
3. **Original Time Window**: Indicates the initially planned time slots for each care.
4. **Treatment Duration**: Time required for each care.
5. **Data Transmission**: Time for administrative updating.

5.2 Model Visits2

The format displayed in the image illustrates the presentation format of the daily nursing home care schedule, adapted to the Visits2 model, which manages specific time constraints.

The main elements presented are the same as with the Visits1 model but with the presence of unavailability windows.

```
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

The Visits1 and Visits2 models have been designed to optimize the scheduling of home-care nurses, taking into account the various complexities and constraints of the home care domain. Each model brings its own strengths and limitations, suited to specific scenarios.

6.1 Model Visits1

The Visits1 model serves as a foundation for efficient planning of nursing treatments, focusing on minimizing working time without incorporating complex constraints such as patient unavailabilities or spacing constraints between treatments. This simplified model has proven effective in scenarios where logistical constraints are reduced and the main objective is to minimize non-productive hours. The Visits1 model is ideal for contexts where patient flexibility is high and treatments are relatively simple to schedule.

6.2 Model Visits2

The Visits2 model, on the other hand, is an evolution of Visits1 that incorporates advanced features such as the management of patient and nurse unavailabilities as well as spacing constraints between treatments. This model is suited to more complex environments where these factors play a crucial role. The Visits2 model has demonstrated its capability to adjust schedules according to multiple constraints, offering more precise planning that conforms to the specific needs of home care.

Testing with different data sets has revealed that Visits2, although effective, may encounter limits when faced with unrealistic travel time estimates, highlighting the importance of using precise and verified data for planning.

6.3 Future Perspectives

For both models, continuous attention to the accuracy of input data, particularly travel times, is crucial. Potential improvements could include integrating more robust techniques to estimate travel times or adjustments in algorithms to better manage uncertainties or variations in travel data.

In summary, Visits1 and Visits2 offer solid frameworks for planning nursing treatments at home, each suited to different levels of complexity in care requirements.