Allocation and scheduling of rescue units

Literature review and applications

Nikola Karagianni
7112112100008

National and Kapodistrian University of Athens
School of Science
Department of Mathematics

Supervisor:
Athanasia Manou

Master Thesis, 2023
Statistics and Operations Research

Three-member committee:

1. Athanasia Manou, Assistant Professor, Athens, 2023 (supervisor)
2. Apostolos Burnetas, Professor, Athens, 2023
3. Antonis Economou, Professor, Athens, 2023

This endeavor would not have been possible without the generous support from the Lilian Voudouri Foundation, who financed me during the academic year 2022-2023.
I am extremely grateful to Athanasia Manou and Fotios Siannis for their advice, guidance, invaluable patience and feedback.

Abstract

This master's thesis explores the optimization of resource allocation in emergency response systems and healthcare facilities, with a focus on the dispatching of rescue units and the scheduling of nurses' appointments.

The first part of the thesis will focus on the allocation and programming of rescue units. The importance of time in emergency situations and the need for quick deployment of rescue units to incident locations is emphasized. To achieve this objective, the thesis draws upon optimization concepts such as the Travelling salesman problem, the multiple Travelling salesman problem, and the vehicle routing problem. These concepts are incorporated into the formulation of a mathematical model that addresses the resource allocation optimization challenges in emergency response scheduling. Subsequently, some heuristic methods will be described to solve the problem, as well as metaheuristic methods to improve the heuristic solution.

The second part of the thesis focuses on a specific problem: the optimization of the scheduling of nurses' appointments for optimum response to medical emergencies. For this, there will be a description of the problem and the model. A new adapted heuristic algorithm is presented as well as a metaheuristic algorithm to improve the heuristic solution. Combining the available theory, artificial intelligence and algorithmic methods, there will be an application on real data.

Finally, the results of the application on real data will be analysed. The performance of the proposed algorithm will be judged on simulated trials.

Περίληψη

Η διπλωματική εργασία εξερευνά τη βελτιστοποίηση κατανομής πόρων σε συστήματα αντιμετώπισης έκτακτης ανάγκης και εγκαταστάσεις υγειονομικής περίθαλψης, με εστίαση στην αποστολή μονάδων διάσωσης και στον προγραμματισμό ραντεβού νοσηλευτών.

Το πρώτο μέρος της εργασίας θα εστιάσει στην κατανομή και τον προγραμματισμό μονάδων διάσωσης. Η σημαντικότητα του χρόνου σε επείγουσες καταστάσεις και η ανάγκη για γρήγορη διάθεση μονάδων διάθεσης σε τοποθεσίες περιστατικών τονίζεται. Για να επιτευχθεί αυτός ο στόχος, η εργασία βασίζεται σε έννοιες βελτιστοποίησης όπως το πρόβλημα του πλανόδιου πωλητή, το πολλαπλό πρόβλημα πλανόδιων πωλητών και το πρόβλημα δρομολόγησης οχήματος. Αυτές οι ιδέες ενσωματώνονται στον σχηματισμό ενός μαθηματικού μοντέλου που απευθύνεται στις προκλήσεις της βελτιστοποίησης κατανομής πόρων. Ακολούθως, θα περιγραφούν κάποιες ευρετικές μέθοδοι για την λύση του προβλήματος, καθώς και μετευρετικές μέθοδοι για την βελτίωση της ευρετικής λύσης.

Το δεύτερο μέρος της εργασίας εστιάζει σε ένα συγκεκριμένο πρόβλημα: την βελτιστοποίηση του προγραμματισμού ραντεβού νοσηλευτών για την βέλτιστη ανταπόκριση σε επείγοντα περιστατικά. Για αυτό, θα υπάρξει μια περιγραφή του προβλήματος και του μοντέλου. Ένας νέος προσαρμοσμένος ευρετικός αλγόριθμος παρουσιάζεται καθώς και ένας μετευρετικός αλγόριθμος για την βελτίωση της ευρετικής λύσης. Συνδυάζοντας την υπάρχουσα θεωρία, τεχνητή νοημοσύνη και αλγοριθμικές μεθόδους, θα υπάρξει εφαρμογή πάνω σε πραγματικά δεδομένα.

Καταληκτικά, τα αποτελέσματα της εφαρμογής πάνω σε πραγματικά δεδομένα θα αναλυθούν. Η απόδοση του προτεινόμενου αλγορίθμου θα κριθεί από προσομοιωμένες δοκιμές.

Contents

Chapter 1

Introduction

In the face of emergencies, time is of the essence. The ability to rapidly deploy rescue units to incident locations can mean the difference between life and loss. Additionally, within the healthcare sector, ensuring that nurses' appointments are optimally scheduled is essential for efficient resource utilization. These challenges demand innovative solutions that leverage optimization techniques to tackle complex routing and scheduling problems. This thesis endeavors to provide such solutions and optimize resource allocation in both emergency response systems and healthcare settings.

1.1 Vehicle Routing Problem

One fundamental problem that underlies the optimization of resource allocation is the Vehicle Routing Problem (VRP). The VRP involves determining the optimal routes for a fleet of vehicles to serve a set of customers, while minimizing total travel distance or time. It is a classic optimization problem with various real-world applications, including emergency response systems and delivery services.

Generally, the VRP is described as the problem of designing optimal delivery or collection routes from one or several depots to a number of geographically scattered cities or customers subject to side constraints.

To be more elaborate, let $V = \{1, \ldots, n\}$ be a set of vertices representing cities with the depot located at vertex 1, and A be the set of arcs. We

denote $G = (V, A)$ as the graph of the cities and the arcs. With every arc (i, j), $i \neq j$, is associated a non-negative distance matrix $C = (c_{ij})$ (i.e. the matrix of travel costs or travel times). When C is symmetrical, it is often convenient to replace A by a set E of undirected edges. In addition, it is assumed that there are m available vehicles based at the depot, where $m_L \leq m \leq m_U$. If $m_L = m_U$, m is said to be fixed. If $m_L = 1$ and $m_U = n - 1$, m is said to be free. If m is not fixed, it is usual to associate a fixed cost f on the use of a vehicle. Moreover, the solution of VRP comes from the designing of a set of least-cost vehicle routes so that:

1. each city in $V \setminus \{1\}$ is visited exactly once by exactly one vehicle;
2. all vehicle routes start and end at the depot;
3. some side constraints are satisfied;

Some other constraints may be:

i. capacity restrictions: a non-negative weight (or demand) d_i is attached to each city $i > 1$ and the sum of weights of any vehicle route may not exceed the vehicle capacity;

ii. the number of cities on any route is bounded above by q (this is a special case of (i) with $d_i = 1$ for all $i > 1$ and $D = q$);

iii. total time restrictions: the length of any route may not exceed a prescribed bound L; this length is made up of intercity travel times c_{ij} and of stopping times δ_i at each city i on the route;

iv. time windows: city i must be visited within the time interval $[a_i, b_i]$ and waiting is allowed at city i;

v. precedence relations between pairs of cities: city i may have to be visited before city j.

1.2 Travelling Salesman Problem

Another related problem that arises in our context is the Travelling Salesman Problem (TSP). The TSP seeks to find the shortest possible route that

a salesman must take to visit a set of cities exactly once and return to the starting city. It is a well-known NP-hard problem with a wide range of applications, including route optimization in emergency response systems and healthcare logistics.

Another definition of the TSP includes graphs. Again, as in the VRP, we have a set of n cities and a set of routes. Therefore, we define the graph $G = (V, A)$. Furthermore, there is a length c_a associated to each route $a \in A$ and the assumption that G is connected. Just as we mentioned before, the goal is to find a trip for the salesman to visit every city requiring the least possible total distance. There is a chance that a route is used more than once, and a city visited more than once. For this reason, each edge has a non-negative weight, in order to avoid tours.

1.3 Multiple Travelling Salesman Problem

In scenarios where multiple agents are involved, such as multiple rescue units or multiple nurses, the Multiple Travelling Salesman Problem (MTSP) becomes relevant. The MTSP extends the TSP by considering multiple salesmen, each with their own set of cities to visit. The objective is to determine optimal routes for each salesman that collectively minimize the overall distance or time.

Here, all salesmen could share the same depot or there could be multiple depots. In all cases, each salesman starts and returns to their depot. If the number of salesmen is not fixed, then a fixed cost could be assigned to each salesman if they are used in the solution. Other variations of the problem could include time windows, i.e. certain cities need to be visited in specific time periods, or even restrictions on the number of cities each salesman can visit. In this problem, aside from the number of cities each salesman will visit, we seek to find the optimal routes in order to minimize the value of the objective function, i.e. distance, time etc.

The MTSP can be considered as a relaxation of the VRP.

1.4 Real-world applications

Theoretically, one could start working on optimizing the schedules of rescue units/nurses using stochastic methods. Although the stochastic vehicle rooting problem introduces a new level of complexity by taking into account uncertainties, such as travel times and changes in demands, it comes with an increase in the complexity making the solution of the problem practically unfeasible. The same applies when we consider the Travelling Salesman Problem (TSP) and the Multiple Travelling Salesman Problem (MTSP). Consequently, many algorithms have been created to approach the optimal solution, while keeping the complexity to a minimum.

All the algorithms formulate the problem within the context of the Vehicle Rooting Problem (VRP). At the same time, they incorporate elements of the Travelling Salesman Problem (TSP) and the Multiple Travelling Salesman Problem (MTSP) into our resource allocation optimization problem. These problems provide insights into the allocation and routing of multiple rescue units or nurses, taking into account their respective tasks and constraints.

It is important to highlight that the results obtained from these algorithms need to be obtained within a logical timeframe. The urgency and time-sensitive nature of emergencies and medical appointments require practical solutions that can be implemented efficiently. Therefore, the focus of this thesis is to strike a balance between achieving accurate results and obtaining them within a reasonable amount of time.

Optimizing the dispatching of rescue units and the scheduling of nurses' appointments can lead to reduced response times during emergencies, enhanced coverage, and increased patient satisfaction. The proposed models and algorithms can serve as valuable tools for decision support systems in real-world emergency response systems and healthcare facilities.

In the first part of this thesis, the emergency dispatching of rescue units is studied and analysed. The units are ranked, their service times may be random variables with a standard deviation, and the problem seeks to minimize a weighted sum ($R/ST_{SD}/\sum w_j C_j$). There is a satisfying number of heuristic algorithms in existence. In order to decide which could be the most

useful in emergency situation, a simulation was designed and the results were compared. Moreover, the results from each algorithm are inserted in a number of metaheuristic algorithms, in order to achieve better results. When we arrive at a decision about the best pair of heuristic-metaheuristic algorithms, we pass on to medical appointments and emergencies.

In the second part, the pair of algorithms is adapted on a given problem and a simulation is run on real data. The schedules then are used to run a simulation on medical emergencies.

In conclusion, this master's thesis aims to contribute to the field of resource allocation optimization in emergency response systems and healthcare settings. By addressing the challenges of emergency response and nurse appointment scheduling, this research strives to provide practical and effective solutions that optimize resource allocation, minimize response times, and enhance the overall effectiveness of emergency response systems and healthcare services.

Chapter 2

Emergency incidents

This chapter focuses on the problem of optimal allocation of rescue units in an emergency situation. Specifically, a study by Wex et al., 2014, is presented.

2.1 Problem specification

Let m be the number of available rescue units and n the number of incidents that need to be processed. For this problem, we assume that $m \leq n$, meaning that the number of available rescue units is smaller than or equal to the number of incidents, which is usually the case in natural disasters. Moreover, we assume the following properties:

Property 1. Each incident has specific requirements and every rescue unit has different capabilities. This property accounts for the fact that not every rescue unit is able to process each incident.

Property 2. The processing times depend both on the incident and on the rescue unit.

Property 3. The travel times between the locations of incidents vary between units.

Property 4. The processing of an incident must not be interrupted (non-preemption).

Property 5. Each incident has a different significance. Therefore, it is assigned a weighting factor accounting for both casualties and damage induced over time. We name this weight "*factor of destruction*" or "*severity level*". The overall harm is measured, as a proxy, by the sum of weighted completion times regarding the processing of incidents.

In the following section, a decision support model is developed. The goal is to find the schedules and assignments of rescue units to incidents that minimize the sum of completion times of incidents weighted by their severity.

2.2 Mathematical Model

Here, we will present an optimization model to find optimal schedules and assignments of rescue units to incidents. The model is presented in a binary quadratic formulation.
Firstly, we use the following notation:

Table 1
Notation used in the mathematical model.

Input parameters	
n	Total number of incidents, with set $I = \{1, \ldots, n\}$
m	Total number of rescue units, with set $K = \{1, \ldots, m\}$
$w_j \in \mathbb{R}^{\geq 0}$	Factor of destruction (*severity level*) of incident j
$p_j^k \in \mathbb{R}^{\geq 0}$	Time required by rescue unit k to process incident j; ∞ if rescue unit k is incapable of addressing incident j
$s_{ij}^k \in \mathbb{R}^{\geq 0}$	Travel time required by rescue unit k to move from incident i to incident j; if $i = 0$ then rescue unit k resides at its depot before travelling to incident j

$cap_{ki} \in \{0,1\}$	1 if rescue unit k is capable of addressing incident i; 0 otherwise
Decision variables	
$X_{ij}^k \in \{0,1\}$	1 if incident i is processed by rescue unit k immediately before processing incident j; 0 otherwise
$Y_{ij}^k \in \{0,1\}$	1 if incident i is processed by rescue unit k (at any time) before processing incident j; 0 otherwise

Secondly, the mathematical model can be written as:

$$\min_{X_{ij}^k, Y_{ij}^k} \sum_{j=1}^{n} w_j \sum_{k=1}^{m} \sum_{i=0}^{n} \left[p_i^k Y_{ij}^k + \left(p_j^k + s_{ij}^k\right) X_{ij}^k + Y_{ij}^k \left(\sum_{l=0}^{n} X_{li}^k s_{li}^k \right) \right] \tag{O}$$

s.t.

$$\sum_{i=0}^{n} \sum_{k=1}^{m} X_{ij}^k = 1, \qquad j = 1, \ldots, n, \tag{C1}$$

$$\sum_{j=1}^{n+1} \sum_{k=1}^{m} X_{ij}^k = 1, \qquad i = 1, \ldots, n, \tag{C2}$$

$$\sum_{j=1}^{n+1} X_{0j}^k = 1, \qquad k = 1, \ldots, m, \tag{C3}$$

$$\sum_{i=0}^{n} X_{i,n+1}^k = 1, \qquad k = 1, \ldots, m, \tag{C4}$$

$$Y_{il}^k + Y_{lj}^k - 1 \leq Y_{ij}^k, \qquad \begin{array}{l} i = 0, \ldots, n; \quad j = 1, \ldots, n+1; \\ k = 1, \ldots, m; \quad l = 1, \ldots, n, \end{array} \tag{C5}$$

$$\sum_{i=0}^{n} X_{il}^k = \sum_{j=1}^{n+1} X_{lj}^k, \qquad l = 1, \ldots, n; \quad k = 1, \ldots, m, \tag{C6}$$

$$X_{ij}^k \leq Y_{ij}^k, \qquad i = 0, \ldots, n; \quad j = 1, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C7}$$

$$Y_{ii}^k = 0, \qquad i = 0, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C8}$$

$$Y_{ij}^k \leq cap_{ki}, \quad i = 1, \ldots, n; \; j = 1, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C9}$$

$$\sum_{l=1}^{n+1} X_{il}^k \geq Y_{ij}^k, \quad i = 0, \ldots, n; \; j = 1, \ldots, n+1; \; k = 1, \ldots, m, \tag{C10}$$

$$\sum_{l=0}^{n} X_{lj}^k \geq Y_{ij}^k, \quad i = 0, \ldots, n; \; j = 1, \ldots, n+1; \; k = 1, \ldots, m, \tag{C11}$$

$$X_{ij}^k, Y_{ij}^k \in \{0, 1\}, \; i = 0, \ldots, n; \; j = 1, \ldots, n+1; \; k = 1, \ldots, m. \tag{C12}$$

Firstly, we have written the objective function (O) which is the weighted sum of completion times over all incidents. We can write the objective function as $\sum_{j=1}^{n} w_j C_j$, where:

$$C_j = \sum_{k=1}^{m} \left(\sum_{i=0}^{n} \left(p_i^k + \sum_{l=0}^{n} X_{li}^k s_{li}^k \right) Y_{ij}^k + \sum_{i=0}^{n} \left(p_j^k + s_{ij}^k \right) X_{ij}^k \right),$$

which is the completion time of incident j. Since only one rescue unit processes the incident j, only one of the addends is non-zero. That is the one that corresponds to the unit k that processes the incident j. We understand that (for the unit k that processes incident j)

$$\sum_{i=0}^{n} \left(p_j^k + s_{ij}^k \right) X_{ij}^k$$

is the processing time of incident j and the travel time to j starting from the incident that was processed immediately before incident j. It is now clear that (for the unit k that processes incident j) the

$$\sum_{i=0}^{n} \left(p_i^k + \sum_{l=0}^{n} X_{li}^k s_{li}^k \right) Y_{ij}^k$$

is the sum of processing times of the incidents that preceded the incident j and the travel times between those incidents.

Then, the restrictions of the problem follow (C1-C11) and, finally, the restriction of non-negativity.

We note that two new incidents have been added. These are fictitious given

by 0 as the starting point (named depot) and $n+1$ as the ending point. Since these incidents do not require any processing time, we fix $p_0^k = p_{n+1}^k = 0$. Moreover, we give each unit k a setup time $s_{0j}^k \geq 0$ to move from its starting point to incident j. Also, we set $s_{j,n+1}^k = 0$ for all rescue units k, because we do not want to take into consideration the travelling times from their last scheduled incident to their ending point and focus only on the minimization of the response time towards incidents. As we have already mentioned, the factor of destruction of incident j is denoted by w_j, meaning the lower the factor of destruction, the less severe the incident.

In order to ensure that there is exactly one incident that is processed immediately before each of the n non-fictitious incidents, we have constraint (C1). In order to ensure that there is exactly one incident that is processed immediately after each of the n non-fictitious incidents, we have constraint (C2). Constraints (C1) and (C2) also ensure that each non-fictitious incident is processed by one rescue unit. Constraint (C3) follows, which guarantees that each rescue unit starts processing the fictitious incident 0 (the depot). Similarly, constraint (C4) ensures that each unit ends processing the fictitious incident $n + 1$. Constraint (C5) accounts for the transitivity in predecessor relationships. This means that if an incident i is processed (not necessarily immediately) before any incident l and the incident l is processed (not necessarily immediately) before an incident j, then the constraint ensures that $Y_{ij}^k = 1$. If an immediate predecessor for a specific incident j exists, there has to be a successor as given by constraint (C6). If an incident i immediately precedes an incident j, then it generally precedes it, as given by constraint (C7). The constraint (C8) prevents a reflexive, direct or indirect predecessor relationship. If a rescue unit k does not have the capability to process an incident i, they should not be assigned to it. Therefore, we have constraint (C9). The constraints (C10) and (C11) together, ensure that if a rescue unit k does not process an incident i before an incident j, $Y_{ij}^k = 0$. To be more precise, the constraints (C10) and (C11) together, set $Y_{ij}^k = 0$ if a rescue unit k does not process either one of the incidents i or j. However, if a unit k processes both i and j, but j is processed before i, the constraints do not set $Y_{ij}^k = 0$ (it can be set to 1 and still satisfy all the constraints), but thanks to the minimization of the objective function, Y_{ij}^k takes the value 0 (because all the parameters and variables are non-negative).

Constraint (C12) makes the model a binary program.

Each feasible solution of the minimization model represents valid schedules and assignments of all rescue units.

2.3 Heuristics for solving the RUASP

Not only is it proved that the Rescue Unit Assignment and Scheduling Problem (RUASP) is NP-hard, but also some practical runtimes where evaluated. More specifically, small up to moderately large instances with $m, n \leq 40$ were used to evaluate the requested times. Using a mixed integer non-linear programming optimizer, the Simple Branch and Bound solver in GAMS, it was found that even small instances cannot be solved optimally in a practically reasonable time. This conclusion was confirmed in interviews with the German Federal Agency of Technical Relief (THW), where it was made clear that decision support in practice cannot take more than 30 minutes. Therefore, several heuristics for solving the RUASP are presented.

Greedy heuristic: This heuristic method is applied in practice in emergency operations centers, usually in a manually-operated and non-automated decision-making process. Here, the incidents are ranked by their level of their severity and are processed from most severe to least severe. It is for this reason that this method is called GREEDY heuristic.

Construction heuristics: Seven heuristics are adapted (Weng et al., 2001) proposed for solving the $R/ST_{SD}/\sum w_j C_j$ scheduling problem. The heuristics are named SCHED1 to SCHED7.

Improvement heuristics: The classical 2-opt and 3-opt exchange procedure are adapted within a single rescue unit (Lin, 1965; Lin & Kernighan, 1973) as well as multi-unit 2-opt and 3-opt, resulting in four heuristics. Moreover, a load balancing heuristic is presented.

GRASP metaheuristics: The improvement heuristics and the construction heuristics, that are mentioned above into GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristics, are amalgamated.

Monte Carlo-based heuristic: So as to account for randomness in the search procedure, we suggest a Monte Carlo-based heuristic.

More details on the heuristics and metaheuristics will follow.
The suggested heuristics, except for the Monte Carlo-based heuristic, can be divided into two categories. One set of 8 construction heuristics, which generate initial feasible solutions of RUASP instances, and a set of 5 improvement heuristics, which iteratively produce new feasible solutions while testing them for local optimality. Every construction heuristic was combined with every improvement heuristic to produce 40 composed heuristics, which are used in the computational experiments.
Now, the construction heuristics as well as improvement heuristics will be analysed. Then, an description of GRASP metaheuristics and the Monte Carlo-based heuristic will follow.

2.4 Construction heuristics

The set of construction heuristics includes the Greedy heuristic and some construction heuristics originating from the scheduling literature.
Let τ_k be the total processing and setup time for unit k in the corresponding loop and let α_k be the last assignment that was taken on by unit k in the corresponding loop. Here, the setup time is the total time needed for a unit to arrive at the current incident. Let, also, $\tilde{p}_i$ be the average processing time of the units that are capable of processing incident i. In the end, each heuristic will return a list, $\sigma = (\sigma_1, \ldots, \sigma_m)$, of the schedules for all m units.

2.4.1 Greedy heuristic

Today, the Greedy heuristic is used in emergency operations centers to model best practice. The idea, here, is to arrange the incidents in descending order of the factor of destruction and then to assign each of those incidents to a unit cleverly chosen. In order to make the assignment of an incident j to a rescue unit k we take under consideration both the assignment history and the updated travel times of each unit. The pseudocode of the Greedy algorithm is as follows:

1: Sort incidents in decreasing order of severity, $w_1 \geq w_2 \geq \ldots \geq w_n$, and set $C \leftarrow \{w_1, \ldots, w_n\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \;\; \alpha_k \leftarrow 0, \;\; \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} \left(\tau_k + s^k_{\alpha_k, i}\right)$ chooses unit with lowest start time.

8: **else**

9: **return** unsuccessful (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i, \;\; \alpha_{unit} \leftarrow i, \;\; \sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

For example, in an arbitrary iteration $i = \iota$, the algorithm takes the following steps:

1. Creates a set $K^* = \{k \in K | cap_{ki} = 1\}$ which contains all the units that are able to process the incident i.
2. If the above set is non-trivial, the algorithm chooses the unit with the lowest start time. If $K^* = \varnothing$, it returns a message of failure and the algorithm stops.
3. If such a unit is found, the algorithm updates τ_{unit} to $\tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i$, α_{unit} to i and σ_{unit} to $\sigma_{unit} \cup \{i\}$.

Evidently, the Greedy algorithm overlooks the possibility that it may not offer the best results by processing the most severe incidents first, since the processing times may also play a crucial role in the decision-making process. It is also easily proven that the Greedy algorithm could easily fail in producing *good* results to an instance of the RUASP. On the other hand, thanks to its simplicity, the results are generated quickly and can be applied without computational support for small instances.

2.4.2 Scheduling heuristics

The algorithms in this subsection consider the trade-off between severity and processing time. This way, 7 heuristics are adapted to fit the $R/ST_{SD}/\sum w_jC_j$ scheduling problem. The initial heuristics were suggested by Weng et al. (2001).

The SCHED1 heuristic that follows, differs from the Greedy algorithm. Firstly, the ratio of the processing time averaged over all units to the severity level is what determines the order of the jobs. Secondly, so as to assign incidents to units, except for the time required to travel to the location of the respective incident, the time required to process the incident is also taken under consideration. The SCHED1 algorithm proceeds as follows:

1: Sort incidents by
$\frac{\tilde{p}_1}{w_1} \leq \frac{\tilde{p}_2}{w_2} \leq \ldots \leq \frac{\tilde{p}_n}{w_n}$ with $\tilde{p}_i \leftarrow \frac{1}{M} \sum_{k\in\{\kappa|cap_{\kappa i}=1\}} p_i^k$, $(M = \#\{\kappa|cap_{\kappa i} = 1\})$
being the average processing time of incident i, and set
$C \leftarrow \{\frac{\tilde{p}_1}{w_1}, \ldots, \frac{\tilde{p}_n}{w_n}\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \;\; \alpha_k \leftarrow 0, \;\; \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K|cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} \left(\tau_k + s^k_{\alpha_k, i} + p^k_i\right)$ chooses unit with lowest sum of time.

8: **else**

9: **return** unsuccessful (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i, \quad \alpha_{unit} \leftarrow i, \quad \sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

The second scheduling heuristc is named SCHED2 and has a slight change from SCHED1. Here, the incident in a arbitrary iteration, is assigned to the rescue unit which has the lowest processing time. To put it simply, the pseudocode of SCHED2 takes the following form:

1: Sort incidents by $\frac{\tilde{p}_1}{w_1} \leq \frac{\tilde{p}_2}{w_2} \leq \ldots \leq \frac{\tilde{p}_n}{w_n}$ with $\tilde{p}_i \leftarrow \frac{1}{M} \sum_{k \in \{\kappa | cap_{\kappa i} = 1\}} p^k_i, \quad (M = \#\{\kappa | cap_{\kappa i} = 1\})$ being the average processing time of incident i, and set $C \leftarrow \{\frac{\tilde{p}_1}{w_1}, \ldots, \frac{\tilde{p}_n}{w_n}\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \quad \alpha_k \leftarrow 0, \quad \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} p_i^k$ chooses unit with lowest processing time.

8: **else**

9: **return** unsuccessfully (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit},i} + p_i^{unit}$, $\alpha_{unit} \leftarrow i$, $\sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

We easily understand that the algorithm of SCHED2 differs from that of SCHED1 only in the 7th step.

The third scheduling heuristic is named SCHED3 and differs from SCHED1 by considering processing times and travel times but ignoring history. Therefore, the SCHED3 algorithm takes the following form:

1: Sort incidents by $\frac{\tilde{p}_1}{w_1} \leq \frac{\tilde{p}_2}{w_2} \leq \ldots \leq \frac{\tilde{p}_n}{w_n}$ with $\tilde{p}_i \leftarrow \frac{1}{M} \sum_{k \in \{\kappa | cap_{\kappa i} = 1\}} p_i^k$, $(M = \#\{\kappa | cap_{\kappa i} = 1\})$ being the average processing time of incident i, and set $C \leftarrow \{\frac{\tilde{p}_1}{w_1}, \ldots, \frac{\tilde{p}_n}{w_n}\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \ \alpha_k \leftarrow 0, \ \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} \left(s^k_{\alpha_k, i} + p^k_i\right)$ chooses unit with lowest sum of travel and processing time.

8: **else**

9: **return** unsuccessful (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i, \quad \alpha_{unit} \leftarrow i, \quad \sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

The next heuristic presented is the SCHED4 heuristic which differs from SCHED1 in the first step. Here, the incidents are renumbered using their minimum processing time. Hence, the SCHED4 algorithm takes the following form:

1: Sort incidents by
$\frac{\tilde{p}_1}{w_1} \leq \frac{\tilde{p}_2}{w_2} \leq \ldots \leq \frac{\tilde{p}_n}{w_n}$ with $\tilde{p}_i \leftarrow \min_{k \in \{\kappa | cap_{\kappa i} = 1\}} p^k_i$
being the minimum processing time of incident i, and set
$C \leftarrow \{\frac{\tilde{p}_1}{w_1}, \ldots, \frac{\tilde{p}_n}{w_n}\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \ \alpha_k \leftarrow 0, \ \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} \left(\tau_k + s^k_{\alpha_k, i} + p^k_i\right)$ chooses unit with lowest sum of start and processing time.

8: **else**

9: **return** unsuccessfully (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i, \quad \alpha_{unit} \leftarrow i, \quad \sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

The next heuristic presented is the SCHED5 heuristic which differs from SCHED2 in the first step. Here, similarly to the previous case, the incidents are renumbered using their minimum processing time. Hence, the SCHED5 algorithm takes the following form:

1: Sort incidents by
$\frac{\tilde{p}_1}{w_1} \leq \frac{\tilde{p}_2}{w_2} \leq \ldots \leq \frac{\tilde{p}_n}{w_n}$ with $\tilde{p}_i \leftarrow \min_{k \in \{\kappa | cap_{\kappa i} = 1\}} p^k_i$
being the minimum processing time of incident i, and set
$C \leftarrow \{\frac{\tilde{p}_1}{w_1}, \ldots, \frac{\tilde{p}_n}{w_n}\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \quad \alpha_k \leftarrow 0, \quad \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} p_i^k$ chooses unit with lowest processing time.

8: **else**

9: **return** unsuccessfully (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit},i} + p_i^{unit}$, $\alpha_{unit} \leftarrow i$, $\sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

The scheduling heuristic SCHED6 differs from SCHED3 in the first step. Here, similarly to the case of SCHED4, the incidents are renumbered using their minimum processing time. Hence, the SCHED6 algorithm takes the following form:

1: Sort incidents by
$\frac{\tilde{p}_1}{w_1} \geq \frac{\tilde{p}_2}{w_2} \geq \ldots \frac{\tilde{p}_n}{w_n}$ with $\tilde{p}_i \leftarrow \min_{k \in \{\kappa | cap_{\kappa i} = 1\}} p_i^k$
being the minimum processing time of incident i, and set
$C \leftarrow \{\frac{\tilde{p}_1}{w_1}, \ldots, \frac{\tilde{p}_n}{w_n}\}$.

2: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \ \alpha_k \leftarrow 0, \ \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

3: **for** $\iota = 1$ **to** n **do**

4: Select incident $i \leftarrow \iota$ to be processed.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident.

6: **if** $K^* \neq \varnothing$ **then**

7: $unit \leftarrow \arg\min_{k \in K^*} \left(s^k_{\alpha_k, i} + p^k_i\right)$ chooses unit with lowest sum of travel and processing time.

8: **else**

9: **return** unsuccessfully (no feasible assignment).

10: **end if**

11: Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i$, $\alpha_{unit} \leftarrow i$, $\sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$.

12: **end for**

13: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

We note that for the last 3 heuristics (SCHED4, SCHED5, SCHED6), it is required that a minimum $\tilde{p}_i \leftarrow \min_{k \in \{\kappa | cap_{\kappa i} = 1\}} p^k_i$ exists. If it doesn't, then the respective incident cannot be processed by any unit. Therefore such an instance has no feasible solution. The following algorithm SCHED7 selects both incident and unit in the same step. This way, drawbacks induced by pre-ordering incidents (as in algorithms SCHED1 to SCHED6) are avoided. The pseudocode of the SCHED7 algorithm is as follows:

1: Initialize the current completion time of each rescue unit, set all rescue units to start at the depot, give an empty set of assignment to each rescue unit i.e.

$$\tau_k \leftarrow 0, \;\; \alpha_k \leftarrow 0, \;\; \sigma_k \leftarrow \varnothing \quad \forall k \in K.$$

2: Initialize list of incidents $I \leftarrow \{1, \ldots, n\}$.

3: Set $C \leftarrow \{\frac{\tau_k + s^k_{\alpha_k,i} + p^k_i}{w_i} | i \in I, k \in K\}$ and $c \leftarrow \min_{i \in I, k \in K} \frac{\tau_k + s^k_{\alpha_k,i} + p^k_i}{w_i}$.

4: **for** $\iota = 1$ **to** n **do**

5: Select incident $i^* \in I$ and unit $k^* \in K$ corresponding to c, i.e. here is the ratio of completion time to severity level minimal. If no minimum exists, stop unsuccessfully (no feasible assignment possible).

6: Update $I \leftarrow I \setminus \{i^*\}$, $\tau_{k^*} \leftarrow \tau_{k^*} + s^{k^*}_{\alpha_{k^*},i^*} + p^{k^*}_{i^*}$, $\alpha_{k^*} \leftarrow i^*$, $\sigma_{k^*} \leftarrow \sigma_{k^*} \cup \{i^*\}$.

7: Update $C \leftarrow \{\frac{\tau_k + s^k_{\alpha_k,i} + p^k_i}{w_i} | i \in I, k \in K\}$ and $c \leftarrow \min_{i \in I, k \in K} \frac{\tau_k + s^k_{\alpha_k,i} + p^k_i}{w_i}$.

8: **end for**

9: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

2.5 Improvement heuristics

In this section, some improvement heuristics for k-opt node exchanges and load balancing are presented. The k-opt node exchanges is used to find a locally optimal solution by exchanging k nodes (incidents), either in the same unit (by exchanging the order in which the unit processes them) or between units. It was found that the 3-opt node exchange had a bigger percentage to arrive at a locally optimal solution than the 2-opt node exchange. The k-opt node exchange for $k \geq 4$ is more time consuming than the 3-opt node exchange, due to the increase in the complexity. It was found, though, that the 4-opt node exchange, despite being more complex, did not have a significantly larger chance of arriving at a local optimal solution than the 3-opt node exchange. For this reason, only the 2-opt and 3-opt node exchange heuristics will be described.

2.5.1 Routing heuristics

In the routing literature, k-opt exchange procedures comprise improvement heuristics that are used for solving the Travelling salesman problem

(Lin, 1965; Lin & Kernighan, 1973), where in each iteration a k-opt exchange is applied until no further k-opt exchange leads to an improvement of the objective value (this means that a local optimum is reached). In the current problem, though, the exchange of 2 or 3 edges across units may lead to infeasible solutions. This could happen if (sequences of) incidents are assigned to units which are not capable of processing these incidents. For this reason, not edges, but nodes (i.e. incidents) are exchanged. These moves are referred to as 2-nodes and 3-nodes exchange respectively. These exchange procedures are applied in two ways. Firstly, a k-node exchange is applied inside the schedule of each rescue unit individually (named 2NSU with $k = 2$ and 3NSU with $k = 3$ respectively). Secondly, exchanges are applied across schedules of multiple rescue units (named 2NMU with $k = 2$ and 3NMU with $k = 3$ respectively). The procedures of the resulting four heuristics are shown below:

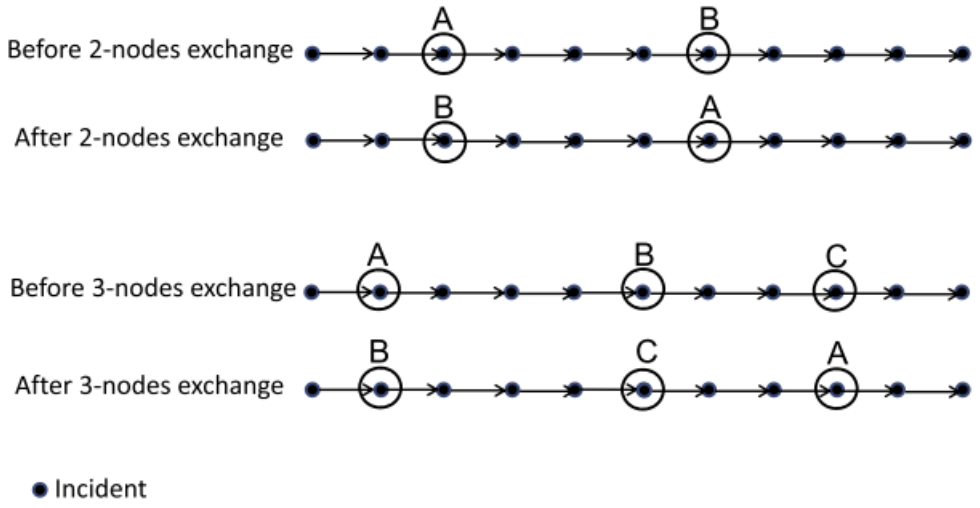

Fig. 2 Illustration of 2-nodes and 3-nodes exchange steps in a single unit.

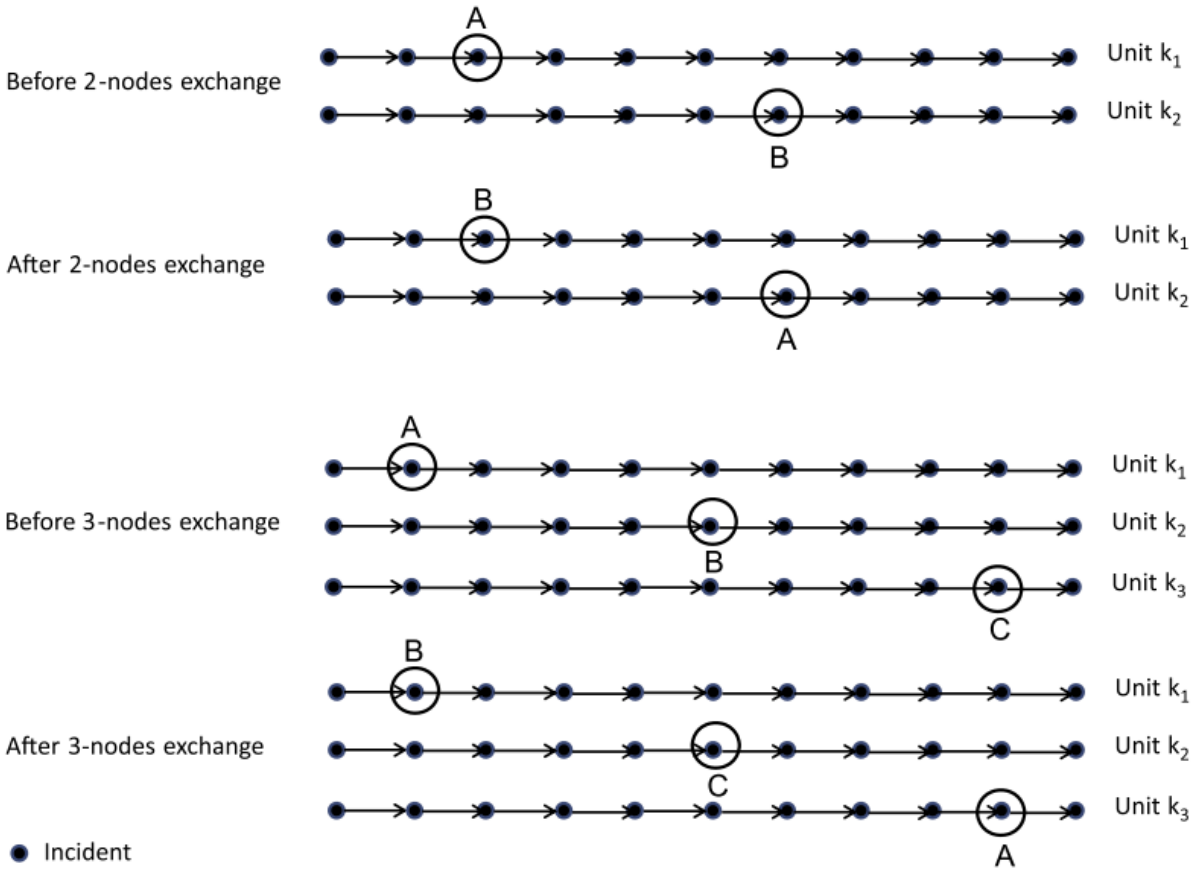

Fig. 3 Illustration of 3-nodes exchange steps across units.

2.5.2 Load balancing heuristic

In this subsection, it is aimed to solve a specific problem. More specifically, when queues of rescue units tend to get long in large-scale disaster scenarios, the process of the incidents at the end of the queue need to wait comparably long. This way, excessively large harm is unavoidable (in terms of objective value). So as to avoid an extremely severe impact, a load balancing heuristic LOADBAL is suggested. This aims to improve a current solution by reassigning the last incidents in a queue to the end of another queue. Let i_k be the last incident in the (ordered) list σ_k. The pseudocode of the LOADBAL heuristic is as follows:

1: Initialize (for $|\sigma_k| = m_k,\ \sigma_k = \{k_1, k_2, \ldots, k_{m_k}\}$)
$\mathrm{harm}(\sigma_k) \leftarrow \sum_{\zeta=1}^{m_k} w_{k_\zeta} \left(\sum_{l=1}^{\zeta} s^k_{k_{l-1},k_l} + p^k_{k_l} \right)$ to be the harm related to unit $k \in K$.

2: **repeat**

3: $k^* \leftarrow \arg\max_{k \in K} \mathrm{harm}(\sigma_k)$ selects the unit k^* with the highest harm and the last incident i_{k^*} processed by this unit .

4: Select the unit k' for which the processing of incident i_{k^*} as the last incident of the queue results in the lowest additional harm, i.e.
$k' \leftarrow \arg\min_{\kappa \in \{k \in K | cap_{\kappa, i_{k^*}} = 1\}} \mathrm{harm}(\sigma_k \cup \{i_{k^*}\}) - \mathrm{harm}(\sigma_k)$.

5: Determine the reduction and the increase of harm caused by moving incident i_{k^*} from the queue of unit k^* to k', i.e.
$\Delta\,\mathrm{harm}_{k^*} \leftarrow \mathrm{harm}(\sigma_{k^*}) - \mathrm{harm}(\sigma_{k^*} \setminus \{i_{k^*}\})$,
$\Delta\,\mathrm{harm}_{k'} \leftarrow \mathrm{harm}(\sigma_{k'} \cup \{i_{k^*}\}) - \mathrm{harm}(\sigma_{k'})$.

6: **if** $\Delta\,\mathrm{harm}_{k^*} - \Delta\,\mathrm{harm}_{k'} > 0$ **then**

7: Create new solution with less harm by setting
$\sigma_{k^*} \leftarrow \sigma_{k^*} \setminus \{i_{k^*}\}$, $\mathrm{harm}(\sigma_{k^*}) \leftarrow \mathrm{harm}(\sigma_{k^*}) - \Delta\,\mathrm{harm}_{k^*}$,
$\sigma_{k'} \leftarrow \sigma_{k'} \cup \{i_{k^*}\}$, $\mathrm{harm}(\sigma_{k'}) \leftarrow \mathrm{harm}(\sigma_{k'}) + \Delta\,\mathrm{harm}_{k'}$.

8: **end if**

9: **until** $\Delta\,\mathrm{harm}_{k^*} - \Delta\,\mathrm{harm}_{k'} \leq 0$

2.6 GRASP metaheuristics

The construction heuristics follow the same search path repeatedly. To cure this defect, GRASP (Greedy Randomized Adaptive Search Procedure) may offer diversity to the solutions of the construction heuristics (Feo & Resende, 1995; Pitsoulis & Resende, 2002; Risende & Ribeiro, 2003). To be more

precise, GRASP is a multi-start metaheuristic for combinatorial problems in which each iteration consists of two phases: construction and local search. In the construction phase, a construction heuristic is used to create feasible solutions, whose neighbourhood is searched using a improvement heuristic. The search continues until it reaches a local minimum. The result is the best overall solution. GRASP variants of algorithms GREEDY and SCHED1 to SCHED7 as construction heuristics are given by the following pseudocode.

1: Initialise $S \leftarrow \emptyset$.

2: **for** iter $= 1$ to $\mathcal{N}$ (max. iterations)

3: Perform greedy randomized construction by initializing candidate set C, i.e. perform initial steps in algorithms GREEDY and SCHED1 to SCHED7 respectively.

4: **for** $l = 1$ **to** n **do**

5: Compute $c_{\min} \leftarrow \min\{c|c \in C\}$ and $c_{\max} \leftarrow \max\{c|c \in C\}$.

6: RLC $\leftarrow \{c \in C|c \leq c_{\min} + \alpha(c_{\max} - c_{\min})\}$.

7: Select randomly a value $c \in$ RLC and let i be the corresponding incident.

8: Perform steps inside the loop in algorithms GREEDY or SCHED1 to SCHED7 without reassigning i.

9: Update $C \leftarrow C \setminus \{i\}$.

10: **end for**

11: Set $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

12: Perform local search upon σ by one of the improvement heuristics giving σ'. Update list of solution by $S \leftarrow S \cup \{\sigma'\}$.

13: **end for**

14: **return** solution $\min S$.

Here, α is the percentage of the number covered by a greedy choice. This kind of candidate list limitation is seen in the bibliography as *value* restriction. Furthermore, limitations on the candidate list can be pursued by allowing only the β best elements, in a type of limitation referred to as a *cardinality* restriction. Of course, one can apply both limitations simultaneously.

To specify, in steps 3 and 8, only one of the algorithms (GREEDY, SCHED1, ..., SCHED7) is used and not any combinations of them. It all depends on which solutions we want to ameliorate by adding randomness to the finding of the solution.

2.7 Monte Carlo-based heuristic

Finally, in order to solve the RUASP, a Monte Carlo-based heuristic is designed. The Monte Carlo simulation offers flexibility in future extensions of the optimization model. For example, it is flexible in co-allocation of rescue units and the consideration of informational uncertainty. Moreover, due to the many constraints of the RUASP, it is obvious that this problem is highly flexible. Regardless, it is expected that the Monte Carlo-based heuristic will not easily get stuck in a local optimum. Of course, there are more complex scenarios where "*evaluation procedures rely a great deal on trial and error*". In contrast, this defect is cured by a Monte Carlo method.

It is denoted that the incidents are iteratively scheduled in two stages. In the first stage, an incident is assigned to one of the D most appropriate rescue units. Here, the appropriateness is defined only based on the processing times of the units. The goal behind this choice, is double. Firstly, assignments of incidents to units that require an extremely long time for processing are avoided (which is why $D \in [0\%, 100\%]$). Secondly, this way, randomness is included. The last is done by not assigning incidents to units that require the shortest processing time among all units. In a second stage, each incident is inserted into the incident queue of the previously selected unit. In order to determine the position of the new incident in the queue, the weighted ratio of the severity of incident w_i and the time $p_i^{k^*}$ it takes the selected rescue unit to process this incident are taken into consideration. Each queue k^* lists its incidents in decreasing order of the values $w_i/p_i^{k^*}$.

Just like all Monte Carlo algorithms, the Monte Carlo-based heuristic runs a fixed number of iterations with the Monte Carlo-based heuristic being the one with the lowest value found in all iterations. The two input parameters required are D and $\mathcal{N}$. The value $D \in [0\%, 100\%]$ is used for the selection of rescue units and $\mathcal{N}$ is the number of the feasible solutions generated. The pseudocode of the Monte Carlo-based heuristic is the following:

1: **for** $iter = 1$ **to** $\mathcal{N}$ (max. iterations) **do**

2: Initialize the cumulative processing time of each rescue unit, rescue units to start at the depot, the ordered list of incidents assigned to unit, i.e. curr_process_time$(k) \leftarrow 0, \quad \alpha_k \leftarrow 0, \quad \sigma_k \leftarrow \emptyset \quad \forall k \in K$.

3: **while** $I \neq \emptyset$ **do**

4: Select next incident $i \in I$ and update $I \leftarrow I \setminus \{i\}$.

5: $K^* \leftarrow \{k \in K | cap_{ki} = 1\}$ are all units capable of processing incident i.

6: **if** $K^* = \emptyset$ **then**

7: **return** unsuccessfully (no feasible assignment).

8: **end if**

9: Sort K^* in ascending order of curr_process_time and select randomly a rescue unit k^* with one of the D lowest values of curr_process_time of all rescue units in K^*.

10: Update $\tau_{k^*} \leftarrow \tau_{k^*} + s^{k^*}_{\alpha_{k^*},i} + p^{k^*}_i, \quad \alpha_{unit} \leftarrow i$.

11: curr_process_time$(k^*) \leftarrow$ curr_process_time$(k^*) + p^{k^*}_i$.

12: Set $\sigma_{k^*} \leftarrow \sigma_{k^*} \cup \{i\}$ and order σ_{k^*} in descending order of $w_i/p^{k^*}_i$.

13: **end while**

14: **end for**

15: **return** $\sigma = (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

2.8 Computational experiments

In this subsection, some computational experiments are presented and the suggested heuristics are evaluated. This evaluation is done mainly based on two touchstones. Firstly, the solutions of the heuristics are compared to a lower bound of the optimal solution. Due to the fact that finding optimal solutions turned out to be computationally infeasible (even for comparatively small instances), lower bounds are needed. So, the gap between the lower bound and a solution found with a heuristic is an upper bound of the gap between the optimal solution and the heuristic solution. This means that the gap underestimates the quality of the heuristic solutions. Secondly, since the GREEDY heuristic represents the best practice behaviour of emergency operation centers, its solution serves as a point of reference. For this reason, the solutions of all the suggested heuristics are evaluated regarding their improvement over the GREEDY heuristic. Subsequently, an appropriate RUASP relaxation will be found so as to estimate lower bounds. Afterwards, the data generation of the experiments are explained and, finally, the results and runtimes are presented and discussed.

2.8.1 Relaxation of the RUASP

An effort was made to solve the binary quadratic programming formulation of the problem using the software package GAMS. Although, due to the NP-hardness of the RUASP, optimal solutions could not be reached even for small instances (40 incidents and 40 rescue units). It is for this reason that appropriate relaxations of the RUASP were derived. It was denoted that even after the relaxation of some of the constraints, runtimes varied between 11 and 22 hours, with an average of 15.6 hours. The trials showed an exponential increase in the runtimes with the increase of numbers of units and incidents, with the number of incidents having a stronger impact. It is obvious that these runtimes to find an optimal solution are inappropriate.

2.8.2 The major earthquake in Japan in 2011

In a numerical experiment, data was drawn from interviews with associates of the THW (Bundesanstalt Technisches Hilfswerk) which were in direct contact to first search and rescue teams after the major earthquake in Japan in 2011.

Accordingly, the values of the input parameters are as described in the Table 2 below:

Table 2
Settings in randomly generated scenarios. Here, $\mathrm{U}(\alpha, \beta, \gamma)$ is the discrete uniform distribution between α and β with step size γ.

Input parameters	Value, range or distribution
Number of rescue units	$m \in \{10, 20, 30, 40\}$
Number of incidents	$n \in \{10, 20, 30, 40\}$
Number of instances	10
Processing times	$p_j^k \sim \mathrm{N}(20, 10)$
Travel times	$s_{ij}^k \sim \mathrm{N}(1, 0.3)$
Factors of destruction	$w_j \in \{1, 2, 3, 4, 5\}$
Capabilities of rescue units	$A_k \sim \mathrm{U}(1, 5, 1),\ k \in K$
Capabilities required by incidents	$R_i \sim \mathrm{U}(1, 5, 1),\ i \in I,$ $cap_{ki} = \begin{cases} 1, & \text{if } A_k = R_i; \\ 0, & \text{else} \end{cases}$
Number of iterations	500000

It is assumed that there is a maximum of 40 units and 40 incidents (Wex et al, 2014, 697-708). Processing times are large compared to travel times. So, travel times should have a smaller mean value as well as variance. Also, the factor of destruction of an incident indicates the level of severity. This resulted in 5 levels of severity: low (1), guarded (2), elevated (3), high (4), and severe (5) harm. The severity levels are selected according to a discrete uniform distribution. The "rescue units" had a total of 5 capabilities (policemen, fire brigades, paramedics, search and rescue units, special casualty access teams). It is also assumed that each incident requires exactly one rescue unit.

2.8.3 Summary of data evaluation

The scenarios considered consisted of 10, 20, 30 and 40 incidents and units, where the number of units did not exceed the number of incidents. For each combination of the number of incidents and the number of units, 10 instances were randomly generated and solved by all heuristics (combined or not with

a metaheuristic). What was measured in the end, was the average ratio $\mu\left(\frac{H_i}{LB}\right)$, where H_i is the heuristic solution and LB is the lower bound. This means that for smaller ratios, better solutions are gained (closer to the lower bound). Moreover, the respective averages where calculated when applying the GREEDY heuristic without any improvement heuristic, which is what represents the best current practice. The experiments showed the following:

- Every combination of construction and improvement heuristic improved the results of the GREEDY algorithm in all instance sizes at a significance level of .01 (p-value of t-test). Even after composing the GREEDY algorithm with any metaheuristic, the results were better than simply applying the GREEDY algorithm (.01 level of significance).

- Results showed that the SCHED7 construction heuristic in combination with any of the metaheuristics lead to superior results in all instances compared to the other construction heuristics combined with the same or any other improvement heuristic (metaheuristic) (.05 significance level with a few exceptions).

- The results of the GREEDY heuristic combined with any of the metaheuristics are worsening as the problems gets larger. On the contrary, the combinations of the rest of the heuristics with any metaheuristics maintain mean ratios below 1.5.

- Concerning the MC heuristic, results are great in scenarios small in size, but they worsen as the numbers of units and incidents get larger. Still, through, in almost all instances the MC algorithm dominates the GREEDY algorithm in most of its combinations (.01 level).

- Among the combinations with improvement heuristics used, the ones combined with 3NMU showed the best results (at the .05 level). Generally, though, the improvement heuristics can ameliorate the solutions of the construction heuristics (in almost 50% of the instances at the .01 level).

- Even though the GRASP metaheuristic showed mixed results, the results achieved by the heuristics ranged at most from 10.9% up to 33.9% above the lowed bound.

2.8.4 Runtimes

As previously mentioned, the results are to be found within minutes in real natural disasters. Thankfully, all the heuristics fulfill this condition. More specifically, all heuristics required less than one second, except for the ones involving the 3NMU which required up to 20 seconds in instances of largest sizes and the MC heuristics. Concerning the MC heuristics, the average runtimes varied between 3.45 minutes (for small instances) and 18.26 minutes (for large instances). Results showed that the runtimes of the MC grow linearly with both rescue units and instances.

Chapter 3

Appointments and emergencies

3.1 Description

In this section, we try to solve a vehicle rooting problem of a fixed number of units responding to medical appointments and emergencies. Firstly, we present the respective properties of the current problem:

Property 1. Each incident has specific requirements and every rescue unit has different capabilities. This property accounts for the fact that not every rescue unit is able to process each incident.

Property 2. The processing times depend only on the incident and not on the rescue unit.

Property 3. The travel times between locations are the same between units.

Property 4. The processing of an incident must not be interrupted (non-preemption).

Property 5. Each incident has the same significance.

Property 6. Each one of the appointments has a time window in which it has to be addressed.

We would prefer to keep the more experienced nurses available to respond to medical emergencies. Consequently, we would prefer to assign as many scheduled appointments as possible to the less experienced nurses and as few as possible to the more experienced ones. Thus, the problem focuses on minimizing the total travel time of the nurses while keeping the more experienced ones available to respond to medical emergencies.

3.2 Available Data

The data comes from a company situated in Peristeri, Greece. This company offers medical services at home in all of Attica, Greece. In total, there are 49 different medical services offered and 12 different nurses that act as independent rescue units. The service times for each of the 49 types of incidents are presented below:

Service no.	Required time	Description (in Greek)
1	5’	Ένεση
2	10’	Αιμοληψία ενηλίκων
3	60’	Αιμοληψία παιδιών
4	12’	Συλλογή ούρων με TIEMAN
5	15’	Λήψη καλλιέργειας αίματος
6	5’	Λήψη καλλιέργειας τραύματος
7	10’	Λήψη καλλιέργειας κοπράνων
8	9’	Λήψη καλλιέργειας πτυελών
9	5’	Λήψη ρινοφαρυγγικού δείγματος
10	40’	Έναρξη ενδοφλέβειας χορήγησης
11	45’	Ενδοφλέβεια χορήγηση (ορός/φάρμακα/παρεντερική)
12	25’	Χορήγηση εντερικής σίτισης
13	30’	Χορήγηση μέσω κεντρικής γραμμής
14	24’	Περιποίηση κεντρικής γραμμής
15	30’	Τοποθέτηση GRIPPER
16	35’	Αλλαγή φλεβοκαθετήρα
17	15’	Έλεγχος φλεβοκαθετήρα
18	15’	Φλασάρισμα - διατήρηση φ/κ
19	30’	Αλλαγή τραχειοστομίας

Service no.	Required time	Description (in Greek)
20	15’	Περιποίηση τραχειοστομίας
21	30’	Αλλαγή γαστροστομίας
22	15’	Περιποίηση γαστροστομίας
23	26’	Αλλαγή και περιποίηση στομίων
24	10’	Οξυγονοθεραπεία
25	20’	Νεφελοποίηση
26	30’	Αναρρόφηση
27	25’	Διαλείπων καθετηριασμός
28	25’	Τοποθέτηση FOLEY
29	27’	Αλλαγή FOLEY
30	32’	Αλλαγή FOLEY υπερηβικού
31	15’	Έλεγχος FOLEY - πλύση κύστεως
32	5’	Αλλαγή ουροσυλλέκτη
33	40’	Τοποθέτηση LEVIN
34	42’	Αλλαγή LEVIN
35	20’	Έλεγχος LEVIN
36	20’	Ηλεκτροκαρδιογράφημα
37	33’	Αέρια αίματος
38	14’	Περιποίηση κατάκλισης - τραύματος
39	40’	Χειρουργικός καθαρισμός
40	20’	Τοποθέτηση αεροστρώματος
41	22’	Κοπή ραμμάτων
42	45’	Ατομική υγιεινή
43	20’	Τοπική υγιεινή
44	30’	Υποκλυσμός υψηλός
45	16’	Υποκλυσμός χαμηλός
46	40	Έναρξη π/κ
47	20’	Περιποίηση κύστης κόκκυγος
48	5’	RAPID/μοριακό test
49	15’	Φλασάρισμα port

Below is presented a table on which we see which services can be provided by each of the 12 units.

Service no.	Unit no.											
	1	2	3	4	5	6	7	8	9	10	11	12
1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	-	✓	-	-	✓	-	-	-	-
4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	✓	✓	✓	✓	✓	-	-	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
11	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
12	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
13	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
14	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
15	✓	✓	✓	✓	✓	✓	-	✓	✓	-	-	-
16	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
17	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
18	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
19	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-
20	✓	✓	✓	✓	✓	✓	-	-	-	✓	-	-
21	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-
22	✓	✓	✓	✓	✓	✓	-	-	-	✓	-	-
23	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
24	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
25	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓
26	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
27	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
28	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
29	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
30	✓	✓	✓	✓	✓	✓	-	✓	-	-	-	-
31	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
32	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
33	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	-
34	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Service no.	Unit no.											
	1	2	3	4	5	6	7	8	9	10	11	12
35	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	-
36	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
37	✓	✓	✓	✓	✓	-	-	✓	-	✓	-	-
38	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
39	✓	✓	-	✓	✓	-	-	-	-	-	-	-
40	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
41	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
42	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
43	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
44	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
45	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
46	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
47	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
48	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
49	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	-	-

Capabilities of nurses

3.3 Mathematical Model

Now, we present an optimization model to find the optimal schedules and assignments of nurses to appointments. The model is presented in a binary linear formulation.
Firstly, we use the following notation:

Table 2
Notation used in the mathematical model.

Input parameters	
n	Total number of incidents, with set $I = \{1, \ldots, n\}$
m	Total number of nurses, with set $K = \{1, \ldots, m\}$
$p_j \in \mathbb{R}^{\geq 0}$	Time required by a capable nurse to process incident j, $j \in I \cup \{n+1\}$, $(p_{n+1} = 0)$

$s_{ij}^k \in \mathbb{R}^{\geq 0}$	Travel time required by nurse k to move from incident i to incident j; if $i = 0$ then nurse k resides at its depot before travelling to incident j; if $j = n + 1$ then nurse k returns to their depot and finishes their shift
$a_k \in \mathbb{R}^{\geq 0}$	The significance of the time of rescue unit k, $k \in K$. The quantities a_k increase as the experience of nurse k increases. This means that if nurse k is more experienced than nurse t, then $a_k \geq a_t$.
$cap_{ki} \in \{0, 1\}$	1 if nurse k is capable of addressing incident i; 0 otherwise
$\min_j \in \mathbb{R}^{\geq 0}$	The earliest moment a nurse can start processing incident j, $j \in I \cup \{n + 1\}$, $(\min_{n+1} = 0)$
$\max_j \in \mathbb{R}^{\geq 0}$	The time by which a nurse must have finished processing incident j, $j \in I$
$\text{Work}_k \in \mathbb{R}^{\geq 0}$	The total number of minutes a nurse has to work, including their break, $k \in K$
$\text{Break}_k \in \mathbb{R}^{\geq 0}$	The number of minutes the break of nurse k lasts, $k \in K$
$B_0^k \in \mathbb{R}^{\geq 0}$	The starting time of nurse k, $k \in K$
$M \in \mathbb{R}^{\geq 0}$	$M > 24 \cdot 60$ (fixed number)
Decision variables	
$X_{ij}^k \in \{0, 1\}$	1 if incident i is processed by nurse k immediately before processing incident j; 0 otherwise, $k \in K$, $i \in \{0\} \cup I$, $j \in I \cup \{n + 1\}$
$Y_{ij}^k \in \{0, 1\}$	1 if incident i is processed by nurse k (at any time) before processing incident j; 0 otherwise, $k \in K$, $i \in \{0\} \cup I$, $j \in I \cup \{n + 1\}$
$B_l^k \in \mathbb{R}^{\geq 0}$	The time by which nurse k will be done with the incident l; M if nurse k doesn't process incident l, $k \in K$, $l \in I \cup \{n + 1\}$.

Notes:

1. For nurse k ($\forall k \in K$), the incidents 0, $n+1$ refer to their depot. Incident 0 refers to the depot as a starting point and incident $n+1$ refers to the depot as a finishing point. Depot for different nurses may differ.

2. For $i, j \in I, \quad i \neq j, \quad s_{ij}^k = s_{ij} \ \forall k \in K$. In other words, between locations of appointments, all nurses need the same transport time. These times may differ only in the case that $i = 0$ or $j = n + 1$, since the depot for different nurses may differ.

3. An incident may include/require more than one services.

4. All times are expressed in minutes.

5. The reader understands that for each incident l, the times B_l^k are calculated taking into account that the nurse cannot start processing the incident l before $\min_l$.

6. The quantity a_k does not depend on the time the shift of nurse k starts. The values of a_k generally depend also on the travel times between locations, as well as the significance of a nurse over the travel time.

Now, the mathematical model can be written as:

$$\min_{X_{ij}^k, Y_{ij}^k} \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{j=1}^{n+1} a_k s_{ij}^k X_{ij}^k \tag{O'}$$

s.t.

$$\sum_{i=0}^{n} \sum_{k=1}^{m} X_{ij}^k = 1, \qquad j = 1, \ldots, n, \tag{C1'}$$

$$\sum_{j=1}^{n+1} \sum_{k=1}^{m} X_{ij}^k = 1, \qquad i = 1, \ldots, n, \tag{C2'}$$

$$\sum_{j=1}^{n+1} X_{0j}^k = 1, \qquad k = 1, \ldots, m, \tag{C3'}$$

$$\sum_{i=0}^{n} X_{i,n+1}^k = 1, \qquad k = 1, \ldots, m, \tag{C4'}$$

$$Y_{il}^k + Y_{lj}^k - 1 \leq Y_{ij}^k, \qquad i = 0, \ldots, n; \quad j = 1, \ldots, n+1; \quad k = 1, \ldots, m; \quad l = 1, \ldots, n, \tag{C5'}$$

$$\sum_{i=0}^{n} X_{il}^k = \sum_{j=1}^{n+1} X_{lj}^k, \qquad l = 1, \ldots, n; \quad k = 1, \ldots, m, \tag{C6'}$$

$$X_{ij}^k \leq Y_{ij}^k, \qquad i = 0, \ldots, n; \quad j = 1, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C7'}$$

$$Y_{ii}^k = 0, \qquad i = 0, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C8'}$$

$$Y_{ij}^k \leq cap_{ki}, \qquad i = 1, \ldots, n; \quad j = 1, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C9'}$$

$$\sum_{l=1}^{n+1} X_{il}^k \geq Y_{ij}^k, \quad i = 0, \ldots, n; \; j = 1, \ldots, n+1; \; k = 1, \ldots, m, \tag{C10'}$$

$$\sum_{l=0}^{n} X_{lj}^k \geq Y_{ij}^k, \quad i = 0, \ldots, n; \; j = 1, \ldots, n+1; \; k = 1, \ldots, m, \tag{C11'}$$

$$B_l^k = \max\left(\sum_{j=0}^{n} \left(B_j^k + s_{jl}^k + p_l\right) X_{jl}^k, \min_l + p_l, \left(1 - Y_{0l}^k\right) M\right), \qquad l = 1, \ldots, n+1; \quad k = 1, \ldots, m, \tag{C12'}$$

$$\sum_{i=0}^{n} \sum_{j=1}^{n+1} (s_{ij}^k + p_j) X_{ij}^k \leq \text{Work}_k - \text{Break}_k, \qquad k = 1, \ldots, m, \tag{C13'}$$

$$B_{n+1}^k \leq B_0^k + \text{Work}_k, \qquad k = 1, \ldots, m, \tag{C14'}$$

$$\max_l \geq \sum_{k=1}^{m} Y_{0l}^k B_l^k, \qquad l = 1, \ldots, n, \tag{C15'}$$

$$X_{ij}^k, Y_{ij}^k \in \{0, 1\}, \quad i = 0, \ldots, n; \; j = 1, \ldots, n+1; \; k = 1, \ldots, m. \tag{C16'}$$

The objective function is the weighted sum of travel times of all nurses. In comparison to the first model we presented, the weights here, a_k, do not correspond to factor of destruction rather to the"cost" of having nurses unavailable for emergencies. The objective function can be written as $\sum_{k=1}^{m} a_k C_k$, where:

$$C_k = \sum_{i=0}^{n} \sum_{j=1}^{n+1} s_{ij}^k X_{ij}^k$$

which is the total travel time of nurse k. If we want to minimize the total time (travel time and service time), we can write the C_k as:

$$C_k = \sum_{i=0}^{n} \sum_{j=1}^{n+1} (p_j + s_{ij}^k) X_{ij}^k.$$

Constraint (C1') ensures that there is exactly one incident that is processed immediately before each one of the n non-fictitious incidents. Constraint (C2') ensures hat there is exactly one incident that is processed immediately after each one of the n non-fictitious incidents. Constraints (C1') and (C2') also ensure that each non-fictitious incident is processed by one nurse. Constraint (C3') ensures that each nurse starts processing the fictitious incident 0 (the depot). Similarly, constraint (C4') ensures that each unit ends processing the fictitious incident $n + 1$. Constraint (C5') accounts for the transitivity in predecessor relationships. This means that if an incident i is processed (not necessarily immediately) before any incident l and the incident l is processed (not necessarily immediately) before an incident j, then the constraint ensures that $Y_{ij}^k = 1$. If an immediate predecessor for a specific incident j exists, there has to be a successor as given by constraint (C6'). If an incident i immediately precedes an incident j, then it generally precedes it, as given by constraint (C7'). The constraint (C8') prevents a reflexive, direct or indirect predecessor relationship. If a rescue unit k does not have the capability to process an incident i, they should not be assigned to it. Therefore, we have constraint (C9'). The constraints (C10') and (C11') together, ensure that if a rescue unit k does not process an incident i before an incident j, $Y_{ij}^k = 0$. Constraint (C12') sets the times B_l^k as described in the notes. Constraints (C13') and (C14') ensure that nurse k does not overwork. More specifically, constraint (C13') ensures that each nurse has time for their break. Constraint (C14') ensures that no nurse will work overtime. Constraint (C15'), ensures that appointment l will be assigned to the nurse that can process it in the required time window.
Constraint (C16') makes the model a binary program.
Each feasible solution of the minimization model represents valid schedules and assignments of all nurses.

In order to find the schedule of each nurse, for example nurse k, we are interested in the quantities $B_l^k < M$, $l \in I$, $k \in K$. Afterwards, so as to estimate the times of arrival of nurse k at the appointments, we calculate the

quantities $A_l^k = B_l^k - p_l$ for the incidents where $B_l^k < M$.

3.4 Proposed Heuristic for the problem

Since the new model to be solved has even more constraints, its solution would take even more time than the previous one. Therefore, we have to find a heuristic algorithm in order to find a heuristic solution in reasonable time. For this reason, and as analysed before, we adapt a modification of SCHED7 heuristic to our problem. The pseudocode of this adapted algorithm is as follows:

1: Initialize the sorted list of incidents $I = \{1, \ldots, n\}$ so that $\min_i \leq \min_{i+1},\ i = 1, \ldots, n-1$.
Let $K = \{1, \ldots, m\}$ be the list of nurses.
Initialize the current completion time of each nurse, set all nurses to start at their depot, give an empty vector of assignments to each nurse i.e.

$$\tau_k \leftarrow B_0^k,\ \ \alpha_k \leftarrow 0,\ \ \sigma_k \leftarrow \{\} \quad \forall k \in K.$$

2: **for** $i = 1$ **to** n **do**

3: We create the set of nurses that can respond to appointment i. Set $C \leftarrow \{(k, A_i^k) | A_i^k = \max\left(\tau_k + s^k_{\alpha_k,i},\ \min_i\right), \text{cap}_{ki} = 1,$
$A_i^k + p_i \leq \min\left(\max_i,\ B_0^k + \text{Work}_k - \text{Break}_k - s^k_{i,n+1}\right),\ k \in K\}$.
If $C = \varnothing$, stop unsuccessfully (no feasible assignment possible).

4: We create the set of nurses that increase the least the value of the objective function.
Set $H \leftarrow \arg\min_k \{a_k s^k_{\alpha_k,i} | \left(k, A_i^k\right) \in C\}$.

5: Choose $k^* = \min\{k | k \in H\}$.

6: Update $\tau_{k^*} \leftarrow A_i^{k^*} + p_i,\ \alpha_{k^*} \leftarrow i,\ \sigma_{k^*} \leftarrow \sigma_{k^*} \cup \{\left(i, A_i^{k^*}\right)\}$.

7: **end for**

8: **return** $\sigma \leftarrow (\sigma_1, \ldots, \sigma_m)$ being the list of schedules.

In step 1, we sort the incidents according to the time their time windows start. Since each time window has the same length, we do not need to take into account the ends of the time windows yet. We also sort the nurses in increasing experience. Afterwards, we do the classic initial step as in SCHED7.
In step 3 (and afterwards), the quantities A_i^k denote the time that nurse k arrives at incident i. For each incident i, the set C contains the couples of nurses that can respond to appointment i and their respectful times of arrival at appointment i. A nurse can respond to an appointment i if they are capable of processing to all the services demanded by appointment i.
In step 4, we create a set H of nurses (that are able to respond to incident i) and increase the least the value of the objective function.
In step 5, we choose one nurse from set H. We can choose randomly, but here we decided to choose the one from the least experienced ones.
In step 6, we update the values of τ_{k^*}, a_{k^*} and the set σ_{k^*} for the chosen nurse k^*.
In step 8, we return the list of schedules.

3.5 Proposed Metaheuristic

In this subsection, we propose an adapted 3NMU metaheuristic algorithm. This adapted algorithm must choose the three nurses and three appointments (one of each) and swap them among the three nurses. Of course, the chosen appointments can only be swapped if each of the nurses can process their newly appointed incidents. The new schedules are created and accepted if the objective function takes a lower value than previously. The pseudocode of this adapted algorithm is as follows:

1: **for** k **in** K **do**

2: **for** l **in** $K \setminus \{k\}$ **do**

3: **for** q **in** $K \setminus \{k, l\}$ **do**

4: **for** $a = 1$ **to** length(σ_k) **do**

5: **for** $b = 1$ **to** length(σ_l) **do**

6: **for** $c = 1$ **to** $\text{length}(\sigma_q)$ **do**

7: It only makes sense to swap appointments that should be addressed in the same time window. So:
if $\min_{\sigma_k(a)[1]} == \min_{\sigma_l(b)[1]} == \min_{\sigma_q(c)[1]}$ &&
$\text{cap}_{k,\sigma_q(c)[1]} == \text{cap}_{l,\sigma_k(a)[1]} == \text{cap}_{q,\sigma_l(b)[1]} == 1$ **do**

8: Set the trial schedules, the "flags" and the current appointment
$\sigma'_k \leftarrow \sigma_k,\ \sigma'_l \leftarrow \sigma_l,\ \sigma'_q \leftarrow \sigma_q,\ \sigma'_k(a) \leftarrow \sigma_q(c),\ \sigma'_l(b) \leftarrow \sigma_k(a),$
$\sigma'_q(c) \leftarrow \sigma_l(b),\ w_k \leftarrow 0,\ w_l \leftarrow 0,\ w_q \leftarrow 0,\ d_1 \leftarrow a,\ d_2 \leftarrow b,\ d_3 \leftarrow c.$

9: Set the time that nurse k finishes with their last appointment
if $d_1 > 1$ **do**

10: $\tau_k \leftarrow \sigma'_k(d_1 - 1)[2] + p_{\sigma'_k(d_1-1)[1]},\ \alpha_k \leftarrow \sigma'_k(d_1 - 1)[1]$

11: **else**

12: $\tau_k \leftarrow B_0^k,\ \alpha_k \leftarrow 0$

13: **end if**

14: Set the time that nurse l finishes with their last appointment
if $d_2 > 1$ **do**

15: $\tau_l \leftarrow \sigma'_l(d_2 - 1)[2] + p_{\sigma'_l(d_2-1)[1]},\ \alpha_l \leftarrow \sigma'_l(d_2 - 1)[1]$

16: **else**

17: $\tau_l \leftarrow B_0^l,\ \alpha_l \leftarrow 0$

18: **end if**

19: Set the time that nurse q finishes with their last appointment
if $d_3 > 1$ **do**

20: $\tau_q \leftarrow \sigma'_q(d_3 - 1)[2] + p_{\sigma'_q(d_3-1)[1]},\ \alpha_q \leftarrow \sigma'_q(d_3 - 1)[1]$

21: **else**

22: $\tau_q \leftarrow B_0^q,\ \alpha_q \leftarrow 0$

23: **end if**

24: Set the new trial schedule of nurse k
while $(w_k == 0 \ \ \&\& \ \ d_1 \leq \text{length}(\sigma'_k))$ **do**

25: Set $A \leftarrow \max\left(\min_{\sigma'_k(d_1)[1]}, \tau_k + s^k_{\alpha_k, \sigma'_k(d_1)[1]}\right)$

26: **if** $A + p_{\sigma'_k(d_1)[1]} < \max_{\sigma'_k(d_1)[1]}$ **do**

27: Set $\tau_k \leftarrow A + p_{\sigma'_k(d_1)[1]},\ \sigma'_k(d_1)[2] \leftarrow A,\ \alpha_k \leftarrow \sigma'_k(d_1)[1],$ $d_1 \leftarrow d_1 + 1$

28: **else**

29: Set $w_k \leftarrow 1$

30: **end if**

31: **end while**

32: **if** $\tau_k + s^k_{\sigma'_k(\text{length}(\sigma'_k))[1], n+1} > \text{Work}_k - \text{Break}_k$ **do**

33: Set $w_k \leftarrow 1$

34: **end if**

35: Set the new trial schedule of nurse l
while $(w_l == 0 \ \ \&\& \ \ d_2 \leq \text{length}(\sigma'_l))$ **do**

36: Set $B \leftarrow \max\left(\min_{\sigma'_l(d_2)[1]}, \tau_l + s^l_{\alpha_l, \sigma'_l(d_2)[1]}\right)$

37: **if** $B + p_{\sigma'_l(d_2)[1]} < \max_{\sigma'_l(d_2)[1]}$ **do**

38: Set $\tau_l \leftarrow B + p_{\sigma'_l(d_2)[1]},\ \sigma'_l(d_2)[2] \leftarrow B,\ \alpha_l \leftarrow \sigma'_l(d_2)[1],$ $d_2 \leftarrow d_2 + 1$

39: **else**

40: Set $w_l \leftarrow 1$

41: **end if**

42: **end while**

43: **if** $\tau_l + s^l_{\sigma'_l(\text{length}(\sigma'_l))[1], n+1} > \text{Work}_l - \text{Break}_l$ **do**

Set $w_l \leftarrow 1$

end if

Set the new trial schedule of nurse q
while $\left(w_q == 0 \;\; \&\& \;\; d_3 \leq \text{length}(\sigma'_q)\right)$ **do**

Set $D \leftarrow \max\left(\min_{\sigma'_q(d_3)[1]}, \tau_q + s^q_{\alpha_q,\sigma'_q(d_3)[1]}\right)$

if $D + p_{\sigma'_q(d_3)[1]} < \max_{\sigma'_q(d_3)[1]}$ **do**

Set $\tau_q \leftarrow B + p_{\sigma'_q(d_3)[1]},\ \sigma'_q(d_3)[2] \leftarrow D,\ \alpha_q \leftarrow \sigma'_q(d_3)[1]$, $d_3 \leftarrow d_3 + 1$

else

Set $w_q \leftarrow 1$

end if

end while

if $\tau_q + s^q_{\sigma'_q(length(\sigma'_q))[1],n+1} > \text{Work}_q - \text{Break}_q$ **do**

Set $w_q \leftarrow 1$

end if

Set trial schedules as new schedules if they lower the value of the objective function
if $w_k + w_l + w_q == 0 \;\; \&\&$
$\sum_{g=k,l,q} a_g \left(\sum_{h=1}^{\text{length}(\sigma'_g)-1} s^g_{\sigma'_g(h)[1],\sigma'_g(h+1)[1]} + s_{0,\sigma'_g(h)[1]g} + s^g_{\sigma'_g(length(\sigma'_g)),n+1}\right) <$
$\sum_{g=k,l,q} a_g \left(\sum_{h=1}^{\text{length}(\sigma_g)-1} s^g_{\sigma_g(h)[1],\sigma_g(h+1)[1]} + s_{0,\sigma_g(h)[1]g} + s^g_{\sigma_g(length(\sigma_g)),n+1}\right)$
do

Set $\sigma_k \leftarrow \sigma'_k,\ \sigma_l \leftarrow \sigma'_l,\ \sigma_q \leftarrow \sigma'_q$

end if

end if

61: **end for**

62: **end for**

63: **end for**

64: **end for**

65: **end for**

66: **end for**

In steps 1-3, we choose 3 different nurses.
In steps 4-6, we choose one appointment from each nurse.
In step 7, we check if these appointments belong in the same time window. If not, these nurses cannot exchange appointments. We also check to see if after the exchange, each of the nurses will be able to process their new appointment. If one of them cannot, they cannot exchange appointments. The "flags" $w_k, \ , w_l, \ , w_q$ reveal if there is a problem with the new assignments.
In step 8, we create the new schedules, with exchanged appointments $(\sigma'_k, \sigma'_l, \sigma'_q)$.
Now, the times of the appointments need correction. For each one of the nurses, we find the first appointment whose time needs correction (d_1, d_2, d_3).
In steps 9-13, we create the current completion time and current position of nurse k.
In steps 14-18, we create the current completion time and current position of nurse l.
In steps 19-23, we create the current completion time and current position of nurse q.
In steps 24-34, we make the corrections on the schedule of nurse k, whilst checking to see if any problems occur.
In steps 35-45, we make the corrections on the schedule of nurse l, whilst checking to see if any problems occur.
In steps 46-56, we make the corrections on the schedule of nurse q, whilst checking to see if any problems occur.
In steps 57-58, if no problems have occurred, we check if the new schedules lower the value of the objective function. If yes, we accept these new assignments.

This algorithm check all possible exchanges between all possible triplets of nurses.

3.6 Simulation

First of all, we divide Attica into the following divisions:

1. Central Athens: This region would encompass the central areas of Athens, including landmarks such as Syntagma Square, Monastiraki, and Plaka.

2. Northern Athens: This region would cover the northern neighborhoods of Athens, including areas like Kifissia, Marousi, and Psychiko.

3. Southern Athens: This region would include the southern neighborhoods of Athens, such as Glyfada, Voula, and Alimos.

4. Western Athens: This region would encompass the western neighborhoods of Athens, including areas like Peristeri, Aigaleo, and Petroupoli.

5. Parnitha Division: This division could include municipalities and areas located in the northern part of Northern West Attica, such as Acharnes, Thrakomakedones, and the vicinity of Mount Parnitha.

6. Mandra Division: This division could encompass municipalities and areas situated in the southern part of Northern West Attica, including Mandra, Elefsina, and the surrounding regions.

7. Lavreotiki Division: This division could include municipalities and areas located in the eastern part of Southern West Attica, such as Lavrio, Sounio, and the surrounding regions.

8. Megara Division: This division could encompass municipalities and areas situated in the western part of Southern West Attica, including Megara, Nea Peramos, and the nearby regions.

9. Marathon Division: This division could include municipalities and areas located in the eastern part of Northern East Attica, such as Marathon, Nea Makri, and the surrounding regions.

10. Rafina-Pikermi Division: This division could encompass municipalities and areas situated in the central part of Northern East Attica, including Rafina, Pikermi, and the nearby regions.

11. Pallini Division: This division could include municipalities and areas located in the western part of Northern East Attica, such as Pallini, Gerakas, and the vicinity.

12. Markopoulo Division: This division could include municipalities and areas located in the eastern part of Central East Attica, such as Markopoulo Mesogeas, Koropi, and the surrounding regions.

13. Paiania Division: This division could encompass municipalities and areas situated in the central part of Central East Attica, including Paiania, Pallini (partially), and the nearby regions.

14. Glyka Nera Division: This division could include municipalities and areas located in the western part of Central East Attica, such as Glyka Nera, Pallini (partially), and the vicinity.

15. Vouliagmeni Division: This division could include municipalities and areas located in the southern part of Southern East Attica, such as Vouliagmeni, Voula, and the surrounding regions.

16. Vari Division: This division could encompass municipalities and areas situated in the central part of Southern East Attica, including Vari, Varkiza, and the nearby regions.

17. Markopoulo Mesogaias Division: This division could include municipalities and areas located in the eastern part of Southern East Attica, such as Markopoulo Mesogaias, Kouvaras, and the vicinity.

18. Piraeus

Then, using Google Maps, we calculate the matrix of times between regions.

Division	1	2	3	4	5	6	7	8	9
1	15	30	30	20	45	35	70	55	70
2	30	20	40	25	25	30	50	40	45
3	30	40	20	50	60	60	60	70	70
4	20	25	50	20	35	30	70	40	55
5	45	25	60	35	15	35	65	50	60
6	35	30	60	30	35	10	75	20	70
7	70	50	60	70	65	75	20	85	80
8	55	40	70	40	50	20	85	10	80
9	70	45	70	55	60	70	80	80	15
10	50	30	50	45	40	50	55	50	30
11	40	15	45	30	30	40	50	40	45
12	50	30	40	40	40	40	40	45	55
13	40	20	40	35	40	35	40	40	50
14	40	20	40	35	35	40	40	45	50
15	45	45	15	40	60	60	55	70	80
16	55	40	25	40	60	60	45	65	80
17	55	30	35	40	55	50	30	55	60
18	30	40	45	25	50	40	85	50	75

Division	10	11	12	13	14	15	16	17	18
1	50	40	50	40	40	45	55	55	30
2	30	15	30	20	20	45	40	30	40
3	50	45	40	40	40	15	25	35	45
4	45	30	40	35	35	40	40	40	25
5	40	30	40	40	35	60	60	55	50
6	50	40	40	35	40	60	60	50	40
7	55	50	40	40	40	55	45	30	85
8	50	40	45	40	45	70	65	55	50
9	30	45	55	50	50	80	80	60	75
10	15	25	30	20	20	55	55	35	55
11	25	10	15	15	15	35	35	20	40
12	30	15	10	15	20	25	20	15	55
13	20	15	15	10	10	40	35	25	45
14	20	15	20	10	10	40	35	25	45
15	55	35	25	40	40	10	10	30	45
16	55	35	20	35	35	10	10	25	55
17	35	20	15	25	25	30	25	10	55
18	55	40	55	45	45	45	55	55	10

From the given data, we know that there are approximately 80 medical appointments per day and $50-55\%$ of those are between the hours 7:00-15:00. We know that in each shift there are 5-6 nurses working. We only know the experience of 12 nurses. Therefore, we will simulate only two shifts, 7:00-15:00 and 15:00-23:00. In each of those shifts, there will be 6 nurses working. In the each shift there will be 40 appointments simulated.

Moreover, from the data, we know that there are 20-25 medical emergencies per day. After we are done with the scheduling using the algorithms, we will simulate 16 medical emergencies, in order to calculate the percentages of the answered and unanswered calls.

We proceed to simulate the medical appointments. For the each shift, there will be 10 appointments per two-hour window. For each incident, in a given time window, we will choose randomly a location. The medical

services demanded for each appointment will be generated with the help of a geometric random variable. From the data, we calculate that a customer asks for approximately 1.25 services. Therefore, we will need a geometric random variable with mean 0.25. We conclude that the random variable will be Geom(0.8) which counts the number of failures before the first success. According to the provided data, 75% of the demanded services are of type (service number) 1, 2, 4, 5, 10, 11, 38. From the rest, 90% of them are of type 7, 8, 14, 23, 24, 26, 28, 29, 32, 34, 44. For this reason, we will choose the types of services according to a vector of probabilities. Each one of the services 1, 2, 4, 5, 10, 11, 38 is chosen with probability $75\%/7 = 75/700$, each one of the services 7, 8, 14, 23, 24, 26, 28, 29, 32, 34, 44 is chosen with a probability $25\% \times 90\%/11 = 225/11000$ and each one of the rest is chosen with a probability $25\% \times 10\%/33 = 25/33000$.

We also assume that we do not have appointments in all regions of Attica each day. Due to the large travel times, unprecedented incidents and medical emergencies, it would be preferable only to visit a few regions each day. In this simulation, we assume that those regions are 1,2,4,18.

We then calculate the levels of experience of each nurse based on how many different types of services they can provide.

We now present the data in R.

```
#The times needed for the provided services

times_of_services=c(5, 10, 60, 12, 15, 5, 10, 9, 5, 40, 45, 25, 30, 24, 30,
35, 15, 15, 30, 15, 30, 15, 26, 10, 20, 30, 25, 25, 27, 32, 15, 5, 40, 42,
20, 20, 33, 14, 40, 20, 22, 45, 20, 30, 16, 40, 20, 5, 15)

#The matrix of capabilities of nurses

Capabilities_of_nurses<- matrix(c(
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
```

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0
), nrow = 49, ncol = 12, byrow = TRUE)

#The sums of rows=the vector of experience

experience_of_nurses=c(49,49,48,48,49,45,36,44,41,43,38,32)

#the travel times of nurses between locations

Travel_times= matrix(c(
15, 30, 30, 20, 45, 35, 70, 55, 70, 50, 40, 50, 40, 40, 45, 55, 55, 30,
30, 20, 40, 25, 25, 30, 50, 40, 45, 30, 15, 30, 20, 20, 45, 40, 30, 40,
30, 40, 20, 50, 60, 60, 60, 70, 70, 50, 45, 40, 40, 40, 15, 25, 35, 45,
20, 25, 50, 20, 35, 30, 70, 40, 55, 45, 30, 40, 35, 35, 40, 40, 40, 25,
45, 25, 60, 35, 15, 35, 65, 50, 60, 40, 30, 40, 40, 35, 60, 60, 55, 50,
35, 30, 60, 30, 35, 10, 75, 20, 70, 50, 40, 40, 35, 40, 60, 60, 50, 40,
70, 50, 60, 70, 65, 75, 20, 85, 80, 55, 50, 40, 40, 40, 55, 45, 30, 85,
55, 40, 70, 40, 50, 20, 85, 10, 80, 50, 40, 45, 40, 45, 70, 65, 55, 50,
70, 45, 70, 55, 60, 70, 80, 80, 15, 30, 45, 55, 50, 50, 80, 80, 60, 75,
50, 30, 50, 45, 40, 50, 55, 50, 30, 15, 25, 30, 20, 20, 55, 55, 35, 55,
40, 15, 45, 30, 30, 40, 50, 40, 45, 25, 10, 15, 15, 15, 35, 35, 20, 40,
50, 30, 40, 40, 40, 40, 40, 45, 55, 30, 15, 10, 15, 20, 25, 20, 15, 55,
40, 20, 40, 35, 40, 35, 40, 40, 50, 20, 15, 15, 10, 10, 40, 35, 25, 45,
40, 20, 40, 35, 35, 40, 40, 45, 50, 20, 15, 20, 10, 10, 40, 35, 25, 45,
45, 45, 15, 40, 60, 60, 55, 70, 80, 55, 35, 25, 40, 40, 10, 10, 30, 45,
55, 40, 25, 40, 60, 60, 45, 65, 80, 55, 35, 20, 35, 35, 10, 10, 25, 55,
55, 30, 35, 40, 55, 50, 30, 55, 60, 35, 20, 15, 25, 25, 30, 25, 10, 55,
30, 40, 45, 25, 50, 40, 85, 50, 75, 55, 40, 55, 45, 45, 45, 55, 55, 10
),nrow = 18, ncol = 18, byrow = TRUE)

We can now start the simulation.
Firstly, we set the values a_k ($\forall k \in K$) as

#The vector of a_k

a=$(10^9, 10^9, 10^8, 10^8, 10^9, 10^7, 10^2, 10^6, 10^4, 10^5, 10^3, 10^1)$

and the locations of the depots of the nurses, for which we use a discrete uniform random variable

#the depots of the nurses

depot=floor(18*runif(12))+1

This way, we get the simulated values:

depot =(11, 18, 7, 12, 12, 15, 14, 4, 1, 3, 13, 14)

We, also, write a function to calculate the travel time of each nurse between locations. For this simulation, we only take into consideration the travel times between appointments, and set travel times from or to depots equal to 0. The function is as follows:

#The travel time of nurse k from location i to location j

S <- function(i,j,k) {

if (i==0 && j!=81) { #from depot to first appointment of nurse k

return(0)

} else if (i==0 && j==81) { #if the nurse isn't assigned to any appointments

return(0)

} else if (i!=0 && j==81) { #from last appointment of nurse k to depot

return(0)

} else { #between appointments

return(Travel_times[i,j])

```
  }
}
```

It is time to simulate the appointments. We will follow the procedure we described:

```
#Appointments

time_of_appointment=matrix(0,nrow=1,ncol=80) #the time needed for
the appointments

location_of_appointment=matrix(0,nrow=1,ncol=80)

appointment_min=matrix(0,nrow=1,ncol=80)

appointment_max=matrix(0,nrow=1,ncol=80)

capability_for_appointment=matrix(0,nrow=12,ncol=80) #this matrix
shows if nurse k (row) can respond to the demands of the appointment
i (column)

for (i in c(1:80)) {

  services=sample(c(1:49), 1+rgeom(1,0.8), prob = c(75/700, 75/700,
25/33000, 75/700, 75/700, 25/33000, 225/11000, 225/11000, 25/33000,
75/700, 75/700, 25/33000, 25/33000, 225/11000, 25/33000, 25/33000,
25/33000, 25/33000, 25/33000, 25/33000, 25/33000, 25/33000, 25/33000,
225/11000, 25/33000, 225/11000, 25/33000, 225/11000, 225/11000, 25/33000,
25/33000, 225/11000, 25/33000, 225/33000, 25/33000, 25/33000, 25/33000,
75/700, 25/33000, 25/33000, 25/33000, 25/33000, 25/33000, 225/11000,
25/33000, 25/33000, 25/33000, 25/33000, 25/33000))

  time_of_appointment[i]=sum(times_of_services[services])

  location_of_appointment[i]=sample(c(1,2,4,18),1)

  for (k in c(1:12)) {

    capability_for_appointment[k,i] = prod(Capabilities_of_nurses[services,k])
```

```
}
#The time window of appointment i
if (i<=10) {
   appointment_min[i]=7*60
   appointment_max[i]=9*60
} else if (i<=20) {
   appointment_min[i]=9*60
   appointment_max[i]=11*60
} else if (i<=30) {
   appointment_min[i]=11*60
   appointment_max[i]=13*60
} else if (i<=40) {
   appointment_min[i]=13*60
   appointment_max[i]=15*60
} else if (i<=50) {
   appointment_min[i]=15*60
   appointment_max[i]=17*60
} else if (i<=60) {
   appointment_min[i]=17*60
   appointment_max[i]=19*60
} else if (i<=70) {
   appointment_min[i]=19*60
   appointment_max[i]=21*60
```

```
    } else if (i<=80) {
        appointment_min[i]=21*60
        appointment_max[i]=23*60
    }
}
```

We gain the following results:

time_of_appointment=(10, 22, 15, 12, 25, 40, 40, 14, 40, 15, 52, 14, 10, 5, 5, 15, 40, 17, 10, 45, 40, 15, 30, 15, 19, 40, 40, 5, 5, 55, 15, 42, 14, 50, 60, 9, 5, 10, 12, 30, 45, 15, 45, 12, 34, 14, 16, 5, 40, 10, 25, 45, 40, 15, 40, 15, 106, 14, 14, 45, 14, 10, 10, 12, 9, 45, 5, 15, 40, 5, 40, 12, 27, 15, 45, 14, 45, 5, 45, 65)

location_of_appointment=(18, 1, 18, 1, 1, 1, 18, 2, 1, 2, 2, 4, 2, 4, 4, 4, 1, 18, 1, 1, 4, 4, 1, 2, 18, 2, 4, 2, 18, 18, 4, 4, 2, 1, 4, 1, 1, 1, 1, 4, 4, 18, 2, 4, 18, 18, 18, 1, 18, 18, 1, 2, 18, 1, 2, 4, 1, 4, 1, 18, 1, 18, 2, 1, 18, 1, 1, 4, 4, 18, 18, 2, 4, 1, 4, 4, 4, 1, 18, 4)

appointment_min=(420, 420, 420, 420, 420, 420, 420, 420, 420, 420, 540, 540, 540, 540, 540, 540, 540, 540, 540, 540, 660, 660, 660, 660, 660, 660, 660, 660, 660, 660, 660, 780, 780, 780, 780, 780, 780, 780, 780, 780, 780, 900, 900, 900, 900, 900, 900, 900, 900, 900, 900, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260)

appointment_max=(540, 540, 540, 540, 540, 540, 540, 540, 540, 540, 660, 660, 660, 660, 660, 660, 660, 660, 660, 660, 780, 780, 780, 780, 780, 780, 780, 780, 780, 780, 900, 900, 900, 900, 900, 900, 900, 900, 900, 900, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1140, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1260, 1380, 1380, 1380, 1380, 1380, 1380, 1380, 1380, 1380, 1380, 1380, 1380)

t(capability_for_appointment)= #the transpose of the matrix, line i describes which nurses can complete the demanded services of appointment i

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

1	1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	1	1	1

We assume that each nurse works for 8 hours= $8 \cdot 60$ minutes, out of which they can have a 30 minute break. We will also assume that the nurses 1,3,5,6,11,12 will work during the first shift and the nurses 2,4,7,8,9,10 will work on the second. This way, the sum of experiences of the nurses on each shift will be equal.
Therefore,

Work=matrix(8*60, 1, 12) #how many minutes each nurse works

Break=matrix(30, 1, 12) #how many minutes lasts the break of each nurse

B0=c(7*60, 15*60, 7*60, 15*60, 7*60, 7*60, 15*60, 15*60, 15*60, 15*60, 7*60, 7*60) #the minute each nurse starts working

Now, we write in code the adapted SCHED7 algorithm.

#Adapted SCHED7 algorithm

#Initialization

t=B0 #current completion time of each nurse

alpha=matrix(0,1,12) #all nurses start at their depot

```
#initiation for the schedules sigma_k, the first column will present the appointment, the second, the nurse that will respond, the third, the estimated time of arrival

my_sigma=matrix(0,nrow=81,ncol=3)

for (i in c(1:80)) {

  C<-matrix(0,ncol=2)

  A=matrix(0,ncol=12)

  for (k in c(1:12)) {

    A[k]=max(t[k]+S(alpha[k],location_of_appointment[i],k), appointment_min[i])

    condition1= (capability_for_appointment[k,i]==1)

    condition2= (A[k] + time_of_appointment[i] < min(appointment_max[i], B0[k] + Work[k]-Break[k] - S(location_of_appointment[i],81,k)))

    if (condition1 && condition2 ) {

      C<-rbind(C,c(k,A[k]))

    }

  }

  if (dim(C)[1]==1) {

    print(paste("No feasible assignment possible for appointment", i))

  } else {

    #create the vector of the nurses that increase the least the value of the objective function

    H=c(a[C[-1,1][1]]*S(alpha[C[-1,1][1]], location_of_appointment[i], C[-1,1][1]))

    for (l in C[-1,1][-1]) {
```

```
    H=c(H,a[l]*S(alpha[l], location_of_appointment[i], l))
  }
  H=C[-1,1][which(H==min(H))]
  #choose which nurse will respond to incident i
  k_star=min(H[which(a[H]==min(a[H]))])
  #Update
  t[k_star]=A[k_star] + time_of_appointment[i]
  alpha[k_star]=location_of_appointment[i]
  my_sigma[i+1]=c(i, k_star, A[k_star]) } }
my_sigma=my_sigma[-1,]
#Then we divide each schedule
sigma_1=my_sigma[my_sigma[,2]==1,-2]
sigma_2=my_sigma[my_sigma[,2]==2,-2]
sigma_3=my_sigma[my_sigma[,2]==3,-2]
sigma_4=my_sigma[my_sigma[,2]==4,-2]
sigma_5=my_sigma[my_sigma[,2]==5,-2]
sigma_6=my_sigma[my_sigma[,2]==6,-2]
sigma_7=my_sigma[my_sigma[,2]==7,-2]
sigma_8=my_sigma[my_sigma[,2]==8,-2]
sigma_9=my_sigma[my_sigma[,2]==9,-2]
sigma_10=my_sigma[my_sigma[,2]==10,-2]
sigma_11=my_sigma[my_sigma[,2]==11,-2]
sigma_12=my_sigma[my_sigma[,2]==12,-2]
```

Each nurse will have their own schedule. Every schedule will be a matrix with two columns. The first column consists of the appointment numbers and the second consists of the expected arrival time of the nurse at the appointment. After we run the algorithm, all appointments have been assigned to nurses. As expected, the nurses with the most experience have the fewest appointments. We now present the schedule of each nurse:

First shift				**Second shift**			
sigma_1=		sigma_11=		sigma_2=		sigma_8=	
5	420	2	420	46	900	44	900
30	660	7	472	80	1260	49	937
		11	552			60	1020
sigma_3=		16	629	sigma_4=		69	1140
4	420	21	664	45	900	77	1260
19	540	23	724	57	1020	78	1325
20	565	33	784	79	1260		
27	660	36	828			sigma_9=	
29	725	37	852	sigma_7=		42	900
34	780			41	900	48	945
38	845	sigma_12=		47	970	53	1020
		1	420	50	996	54	1090
sigma_5=		8	470	51	1036	58	1125
6	420	10	504	52	1091	63	1164
39	780	12	544	61	1166	64	1204
40	812	13	583	62	1210	67	1231
		14	618	65	1230	72	1266
sigma_6=		15	643	70	1249	73	1303
3	420	22	668	71	1264		
9	465	24	708	76	1329	sigma_10=	
17	540	28	743			43	900
18	610	31	780			55	1020
25	660	32	815			56	1085
26	719					59	1120
35	784					66	1149
						68	1214
						74	1260
						75	1295

We move on to simulate 16 medical emergencies. Since we do not know anything about their distribution during the day, we are going to simulate 8 medical emergencies during the first shift and 8 during the second. Once a medical emergency is announced (called in), it must be addressed within 1-2 hours. Each time of announcement will follow a uniform distribution in the respective shift. The time window of the medical emergency starts at the time of the announcement and (for the simulation) ends 90 minutes afterwards. The medical emergencies can demand any type of service. Here, we assume that the number of services follows the same distribution as the number of services of each appointment. Finally, the emergency appointment can be demanded at any region of Attica.

```
#Emergencies

time_of_emergency=matrix(0,nrow=1,ncol=16) #the time needed for the emergencies

location_of_emergency=matrix(0,nrow=1,ncol=16)

emergency_min=matrix(0,nrow=1,ncol=16) #beginning of time window of the medical emergencies

emergency_max=matrix(0,nrow=1,ncol=16) #end of time window of the medical emergencies

capability_for_emergency=matrix(0,nrow=12,ncol=16) #this matrix show if nurse k can respond to the demands of the emergency i

for (i in c(1:16)) {

   emergency_services=sample(c(1:49),1+rgeom(1,0.8))

   time_of_emergency[i]=sum(times_of_services[emergency_services])

   location_of_emergency[i]=sample(c(1:18),1)

   for (k in c(1:12)) {

      capability_for_emergency[k,i] =
prod(Capabilities_of_nurses[emergency_services,k])

   }
```

```
    if (i<=8) { #the emergencies of the first shift
       emergency_min[i]=floor((8*60-30)*runif(1))+7*60
       emergency_max[i]=emergency_min[i]+90
    } else { #the emergencies of the second shift
       emergency_min[i]=floor((8*60-30)*runif(1))+15*60
       emergency_max[i]=emergency_min[i]+90
    }
}
```

We present the results of the algorithm.
The time needed at each of the appointments is:

time_of_emergency= (48, 24, 5, 10, 40, 26, 24, 25, 110, 5, 40, 5, 67, 55, 26, 15)

The beginning of each time window is:

emergency_min= (568, 790, 654, 690, 590, 458, 831, 595, 1302, 955, 924, 1253, 1190, 1188, 1072, 1101)

We are going to address these incidents to nurses 1, 5 (for the first shift) and nurses 2,4 (for the second shift). Since all times are uniformly distributed and the chosen nurses have very few appointments, we only want to calculate the free time of each nurse and the time needed for the emergencies in each shift. For each shift, we will calculate how many minutes are needed to address all emergencies (travel times and times for each emergency). If the free time of the experienced nurses (in each) shift exceeds the time needed for the emergencies, then all emergencies can be addressed. For this part, we do not need to find a complex algorithm that takes into consideration the capabilities of each nurse, because the experienced nurses can address all types of incidents. Therefore, we calculate:

```
emergencies1=time_of_emergency[1] #the total time needed for the emergencies of the first shift (initialization)

emergencies2=time_of_emergency[9] #the total time needed for the emergencies of the second shift (initialization)

#the total time needed for the emergencies of the first shift

for (i in c(2:8)) {

   emergencies1=emergencies1+S(i-1,i,1)+time_of_emergency[i]

}

#the total time needed for the emergencies of the second shift

for (i in c(10:16)) {

   emergencies2=emergencies2+S(i-1,i,1)+time_of_emergency[i]

}

#the free time of nurse 1

if (length(sigma_1)>2) {

   free_time1=8*60-sum(time_of_appointment[sigma_1[,1]])

   for (i in c(2:length(sigma_1[,1]))) {

      free_time1=free_time1-S(i-1,i,2)

   }

} else {free_time1=8*60-sum(time_of_appointment[sigma_1[1]])}

#the free time of nurse 5

if (length(sigma_5)>2) {

   free_time5=8*60-sum(time_of_appointment[sigma_5[,1]])

   for (i in c(2:length(sigma_5[,1]))) {

      free_time5=free_time5-S(i-1,i,2)
```

```
   }
} else {free_time5=8*60-sum(time_of_appointment[sigma_5[1]])}
#the free time of nurse 2
if (length(sigma_2)>2) {
   free_time2=8*60-sum(time_of_appointment[sigma_2[,1]])
   for (i in c(2:length(sigma_2[,1]))) {
      free_time2=free_time2-S(i-1,i,2)
   }
} else {free_time2=8*60-sum(time_of_appointment[sigma_2[1]])}
#the free time of nurse 4
if (length(sigma_4)>2) {
   free_time4=8*60-sum(time_of_appointment[sigma_4[,1]])
   for (i in c(2:length(sigma_4[,1]))) {
      free_time4=free_time4-S(i-1,i,2)
   }
} else {free_time4=8*60-sum(time_of_appointment[sigma_4[1]])}
#we check if the experienced nurses can make it to all the appointments
if (emergencies1 > free_time1 + free_time5 | emergencies2 > free_time2
+ free_time4) {
   print("Not all emergencies can be addressed.")
} else {print("Success.")}
```

And the result is: "Success."

We then run 10000 trials. In each trial, we assumed new depots, new appointments, new medical emergencies the same way as in the simulation above. We calculated the percentage of appointments that cannot be assigned to any nurse via the algorithm and the percentage of medical emergencies that cannot be addressed. As for the medical emergencies, we assume that all will be addressed if the experienced nurses have more combined free time than the sum of travel times between the emergencies and service times for all medical emergencies. If not, then we decrease the demanded time by one medical emergency and check again. We repeat this process and count the failures in addressing medical emergencies until the condition is satisfied. In order to count these failures, we added the following piece of code:

```
if (emergencies1 > free_time1 + free_time5 | emergencies2 > free_time2
+ free_time4) { #if there is not enough time to face all the medical
emergencies

  for (i in c(1:8)) {

    if(emergencies1 > free_time1 + free_time5) { #if there is not
enough time in the first shift

      emergencies1=emergencies1-time_of_emergency[i] #we abstract
the time of one medical emergency

      fails = fails+1 #we increase the number of failed responses by
one

      if (emergencies1 > 0) {

        emergencies1 = emergencies1 - S(i,i+1,1) #we abstract the
travel time to this medical emergency

      }

    }

    if(emergencies2 > free_time2 + free_time4) { #if there is not
enough time in the second shift
```

```
            emergencies2=emergencies2 - time_of_emergency[i+8] #we ab-
stract the time of one medical emergency

            fails = fails+1 #we increase the number of failed responses by
one

            if (emergencies2 > 0) {

                emergencies2 = emergencies2 - S(i,i+1,1) #we abstract the
travel time to this medical emergency

            }

        }

    }

}
```

The results showed that only 0.8% of appointments and 2.59% of medical emergencies cannot be addressed. These failures in scheduling happen in cases where many appointments demand services that only the experienced nurses can provide.

In the case where $a = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$, in 10000 new trials, 0.9% of appointments and 93.7% of medical emergencies could not be addressed. These failures happen because of the values of the vector a. Since we do not assign greater values to the experienced nurses, the appointments are assigned to all nurses without preference, thus leaving the experienced nurses with insufficient free time.

In the case where $a = (3^9, 3^9, 3^8, 3^8, 3^9, 3^7, 3^2, 3^6, 3^4, 3^5, 3^3, 3^1)$, in 10000 new trials, 0.8% of appointments and 2.7% of medical emergencies could not be addressed.

According to these results, it is evident that we have to assign bigger coefficients to more experienced nurses in order for them to have the necessary time to respond to medical emergencies.

In a new scenario, where each customer chooses a service using a discrete uniform over all types of services, we run 10000 trials for the same cases of the value of vector a.

In the case where $a = (10^9, 10^9, 10^8, 10^8, 10^9, 10^7, 10^2, 10^6, 10^4, 10^5, 10^3, 10^1)$, we run 10000 trials and the results showed that 1.8% of appointments and 12.1% of medical emergencies could not be addressed.

In the case where $a = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$, we run 10000 trials and the results showed that 4.1% of appointments and 98% of medical emergencies could not be addressed.

In the case where $a = (3^9, 3^9, 3^8, 3^8, 3^9, 3^7, 3^2, 3^6, 3^4, 3^5, 3^3, 3^1)$, we run 10000 trials and the results showed that 1.8% of appointments and 12.3% of medical emergencies could not be addressed.

From the results we obtained, we deduce that, in all scenarios, we need to assign bigger coefficients to experienced nursed in order to get the best results possible.

Chapter 4

Conclusion

In this master's thesis, we have undertaken a comprehensive examination of the resource allocation optimization challenges present in emergency response systems and healthcare settings. Our primary focus was to address the efficient deployment of rescue units to incident locations and the optimal scheduling of nurses' appointments. Through the application of optimization techniques and algorithmic approaches, our aim was to provide innovative solutions that optimize resource allocation, minimize response times, and enhance the overall effectiveness of emergency response systems and healthcare services.

Throughout the course of this research, we have recognized the critical role that time plays in emergency situations. The ability to swiftly deploy rescue units to incident locations can often be the determining factor between life and loss. Similarly, in healthcare settings, the efficient scheduling of nurses' appointments is paramount for maximizing resource utilization and ensuring timely medical care for patients. These challenges necessitated the development of novel approaches and optimization models capable of addressing the complexities inherent in routing and scheduling problems in both domains.

All the algorithms presented in the first part formulate the first problem within the context of the Vehicle Rooting Problem (VRP). At the same time, they incorporate elements of the Travelling Salesman Problem (TSP) and the Multiple Travelling Salesman Problem (MTSP) into our resource allocation optimization problem. These problems provide insights into the

allocation and routing of multiple rescue units or nurses, taking into account their respective tasks and constraints.

The algorithm presented in the second part of the thesis builds upon the foundation of the SCHED7 algorithm and exhibits remarkable performance, providing results within approximately one second. The algorithm's low failure rates attest to its efficacy. Furthermore, the algorithm's speed allows for the verification of proposed schedules prior to appointment scheduling, thereby minimizing the risk of failures in responding to medical emergencies. Additionally, we propose that the assignment of each medical emergency should be determined in real time, considering the presence of unknown parameters such as increased traffic and potential delays from previous appointments. Reviewing the current status of each nurse before making an assignment would ensure the most appropriate allocation of resources.

The outcomes of this research hold substantial implications for the field of emergency response systems and healthcare resource management. By optimizing the dispatching of rescue units and scheduling of nurses' appointments, the proposed algorithm contributes to the enhancement of emergency response capabilities, improved patient care, and increased operational efficiency. These findings have practical applications for decision support systems within real-world emergency response systems and healthcare facilities.

As avenues for further research, it would be valuable to explore the scalability and robustness of the algorithm in larger-scale scenarios with diverse constraints. Additionally, integrating real-time data and advanced predictive analytics could enhance the algorithm's adaptability to dynamic changes and improve its decision-making capabilities. Further investigations into incorporating additional optimization techniques and considering multi-objective optimization would expand the algorithm's applicability and potential benefits.

In conclusion, this master's thesis has addressed the resource allocation optimization challenges in emergency response systems and healthcare settings. By developing an algorithm that optimizes the dispatching of rescue units and scheduling of nurses' appointments, we have made significant contributions to improving emergency response capabilities, patient care, and

operational efficiency. The findings of this research offer practical implications for decision support systems in real-world emergency response systems and healthcare facilities. Future research endeavors can build upon these findings by exploring scalability, incorporating real-time data, and expanding the algorithm's optimization capabilities.

Bibliography

[1] Bektas, T. (2006). *The multiple traveling salesman problem: an overview of formulations and solution procedures*. Omega, 34, 209–219.

[2] Feo, T. A., & Resende, M. G. C. (1989). *A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem*. Operations Research Letters, 8, 67-71.

[3] Feo, T. A., & Resende, M. G. C. (1995). *Greedy randomized adaptive search procedures*. Journal of Global Optimization, 6, 109-133.

[4] Lin, S. (1965). *Computer solutions of the travelling salesman problem*. Bell System Technical Journal, 44, 2245-2269.

[5] Lin, S., & Kernighan, B. (1973). *An effective Heuristic algorithm for the travelling salesman problem*. Operations Research, 21, 498-516.

[6] Jünger et al (1995). *The Traveling Salesman Problem*. M.O. Ball et al., Eds., Handbooks in OR MS, VoL 7, Chapter 4, 225-330.

[7] Laporte, G. (1992). *The Vehicle Routing Problem: An overview of exact and approximate algorithms*. European Journal of Operational Research, 59, 345-358.

[8] Oyola et al (2018). *The stochastic vehicle routing problem, a literature review, part I: models*. EURO J Transp Logist, 7, 193–221.

[9] Pitsoulis, L., & Resende, M. G. C. (2002). *Greedy randomized adaptive search procedures*. In P. M. Pardalos & M. G. C. Resende (Eds.), Handbook of applied optimization (pp. 168-181). New York: Oxford University Press.

[10] Resende, M. G. C., & Ribeiro , C. C., (2003). *Greedy randomized adaptive search procedures*. In: F. Glover & G. A. Kochenberger (Eds.), Handbook of metaheuristics. US international series in operations research & management science (Vol. 57, pp. 219-249). Boston: Springer.

[11] Weng, M. X., Lu, J., & Ren, H. (2001). *Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective*. International Journal of Production Economics, 70, 215-226.

[12] Wex et al (2014) *Emergency response in natural disaster management: Allocation and scheduling of rescue units*. European Journal of Operational Research, 697-708.