# Metaheuristics for the consistent nurse scheduling and routing problem.

Thomas Macdonald
supervisors: Karl Dörner and Xavier Gandibleux

Department of Production and Logistics
University of Vienna
Bruenner Strasse 72
1210 Vienna – Austria
thomas.macdonald@etu.univ-nantes.fr

**Abstract.** We present a metaheuristic based on local search for the scheduling and routing of mobile nurses. Mobile nurses visit patients at home on a regular basis. Some patients require up to 3 visits per day on predefined time windows. The visits require different qualifications and the nurses have different qualification levels. Downgrading is possible. In this problem besides the cost the service quality is also of major importance. One aspect of service quality is to minimize the number of different nurses that visit one customer. This leads to the consistent routing and scheduling of mobile nurses.

We apply the developed algorithm to standard benchmark instances based on the consistent VRP instances introduced by Groër et al [11].

**Keywords:** Mobile Healthcare, ConVRP, Metaheuristics

## 1 Introduction

Although the traditional vehicle routing problem, or VRP, has been extensively studied by various researchers over the last 50 years, the focus has mostly been on the routing side of the problem, ie how to minimise the total cost or distance traveled by a fleet of vehicles over a set of routes. In this instance the focus is not

so much on this side of the problem, but rather on what Groër et al [11] term the consistency constraint. This is when one constrains the problem in such a way as to try and minimize the number of different vehicles that visit a given client. This is a problem that has only really been studied for the last five years or so.

In this paper, we examine a problem coming from a large Austrian charitable organisation, which employs teams of nurses to visit outpatients at their homes. This is, on the face of it a classical VRP. However, there are several added constraints. First of all the requests from the clients involve time windows. Also there is no demand, and no capacity, but there is a service time constraint. Each task requested by the patient and serviced by a mobile nurse is expected to take a certain amount of time. Also there is a scheduling aspect to the problem, as the demands span a certain amount of days.

Although there is a routing aspect to the problem, the organisation is trying to afford to their patients the best possible service quality. As the patients need medical attention, this objective is very well served by allowing nurses and patients to build a friendly relationship, and so we try to ensure that each patient only ever sees one nurse, insofar as this is possible. This is obviously the exact same thing as was dicussed by Groër et al, ie the consistency issue. Therefore, over the course of this research, we have made it a priority to construct solutions that are good from a consistency point of view.

### 1.1 General Description

The overall objective (see 1 on page 5) is to service a set of clients with a set of nurses who start from home . The main problem is that each client should be visited always by the same nurse. Each client puts in service requests, each of which has a time window and a day associated to it. A shift for a nurse is at most 8 hours long, which defines the maximum route length. Also each nurse

has a skill level, from carer to doctor. Each request has a skill level demand, and upgrading (ie a request for a procedure necessitating a skill level x being serviced by someone with training of level y where $y < x$) is not possible, although downgrading is.

For each client there can be up to 20 service requests per week. Each service request includes a time window, of which there are 5 possible: Early Morning (6am to 8am), Late Morning (8am to 11am), Midday (11am to 1pm), Afternoon (1pm to 4pm), Evening (4pm to 8pm). If a nurse arrives too early at a client's home, they must wait, because the time windows are hard constraints. A week starts on Monday and finishes on Sunday. Each service request involves a specific day of the week. There is also an exact time, which indicates a preference, and is therefore not a hard constraint. Also there is the length of time the task in question will take.

When it comes to skill levels, each nurse has a certain level of qualification. They can be trained to clean, to visit, as a home helper, a carer, or as a doctor. Each service request specifies which level of skill is needed. Upgrading is not possible, so a doctor can service a home help request, but a home helper cannot service a care request.

Each nurse has a maximum number of hours that they can work in the week, in addition to a maximum shift time per day. This factor limits the size the routes can take.

### 1.2 Problem description

We start with several sets:

- $V$ is a set of service requests.
- $V'$ is the set of all nodes (service request locations and depots).
- For each client $c$, we have $V_c$, the set of service requests $c$ has put in.

- $C$ is the set of clients, $K$ is the set of employees (nurses).
- $H$ is the set of days. Monday to Friday would for example be 0 to 4.

Next, there are 3 binary variables, $x_{ij}^{kh}$, $y_i^k$, and $f_c^k$:

- If arc $(i, j)$ is traversed by nurse $k$ on day $h$, then $x_{ij}^{kh} = 0$, 1 otherwise.
- If service request $i$ is assigned to nurse $k$, then $y_i^k = 0$, 1 otherwise.
- If client $c$ is visited by nurse $k$, then $f_c^k = 0$, 1 otherwise.

There are also some real variables.

- $F \geq 0$ is the maximum number of nurses assigned to a client.
- $B_i \geq 0$ is the beginning of service of request $i$.
- $B_{o(k,h)} \geq 0$ is the time of departure from the start depot of nurse $k$ on day $h$.
- $B_{d(k,h)} \geq 0$ is the time of arrival at the end depot of nurse $k$ on day $h$.

The data is modeled as follows:

- $e_i$ is the start of the time window of service request $i$.
- $l_i$ is the end of the time window of service request $i$.
- $s_i$ is the service time of request $i$.
- $T_{(k,h)}$ is the maximum route duration of nurse $k$ on day $h$.
- $l_i$ is the skill level required by service request $i$.
- $q_k$ the skill level provided by nurse $k$.
- $t_{ij}$ is the distance, or travel time from $i$ to $j$.
- $c_{ij}$ is the cost of the travel.

This leads us to the following formulation:

**Formulation**

$$\min \omega F + \sum_{i \in V'} \sum_{j \in V'} \sum_{k \in K} \sum_{h \in H} x_{ij}^{kh} c_{ij} \tag{1}$$

$$\sum_{h \in H} \sum_{k \in K} \sum_{j \in V'} x_{ij}^{kh} = 1 \qquad \forall i \in V, \tag{2}$$

$$\sum_{j \in V'} x_{ij}^{kh} - \sum_{j \in V'} x_{ji}^{kh} = 0 \qquad \forall i \in V, h \in H, k \in K, \tag{3}$$

$$\sum_{j \in V'} x_{o(k,h),j} = 1 \qquad \forall h \in H, k \in K, \tag{4}$$

$$\sum_{i \in V'} x_{i,d(k,h)} = 1 \qquad \forall h \in H, k \in K, \tag{5}$$

$$B_j \geq (B_i + s_i + t_{ij}) x_{ij}^{kh} \qquad \forall i \in V', j \in V', k \in K, h \in H, \tag{6}$$

$$e_i \leq B_i \leq l_i \qquad \forall i \in V, \tag{7}$$

$$B_{d(k,h)} - B_{o(k,h)} \leq T_{(k,h)} \qquad \forall h \in H, k \in K, \tag{8}$$

$$y_i^k \geq \sum_{h \in H} \sum_{j \in V'} x_{ij}^{kh} \qquad \forall i \in V, k \in K, \tag{9}$$

$$y_i^k l_i \leq q_k \qquad \forall i \in V, k \in K, \tag{10}$$

$$|V_c| f_c^k \geq \sum_{i \in V_c} y_i^k \qquad \forall c \in C, k \in K \tag{11}$$

$$F \geq \sum_{k \in K} f_c^k \qquad \forall c \in C, \tag{12}$$

$$x_{ij}^{kh} \in \{0, 1\} \qquad \forall i \in V', j \in V', k \in K, h \in H, \tag{13}$$

$$y_i^k \in \{0, 1\} \qquad \forall i \in V, k \in K, \tag{14}$$

$$f_c^k \in \{0, 1\} \qquad \forall c \in C, k \in K, \tag{15}$$

$$F \geq 0, \tag{16}$$

$$B_i \geq 0 \qquad \forall i \in V'. \tag{17}$$

**Constraints**

Objective function ( 1 on the previous page): The weighting parameter $\omega$ has to be set such that more weight is put on consistency than on routing.

Constraint ( 2 on the preceding page): every service request has to be visited.

Constraint (3): every service request has to be entered and left.

Constraint (4): every nurse has to leave her start depot on every day.

Constraint (5): every nurse has to return to her start depot on every day.

Constraint (6): sets the beginning of service at each location.

Constraint (7): time windows.

Constraint (8): maximum route duration constraints.

(Each nurse has an 8h shift per day)

Constraint (9): request to nurse assignment.

Constraint (10): skill level requirements.

Constraint (11): if nurse $i$ visits client $c$

Constraint (12): the maximum number of nurses assigned to some client.

## 2 State of the art

Although the VRP has been studied extensively over the last 50 years, the focus has always been on the routing aspect of the problem. The type of problem that is dealt with in this paper involves what Groër et al [11] term "consistency requirements". This is where the focus is on a patient as far as possible only being visited by the same nurse. As far as we know this is the only paper that deals specifically with this type of problem. We have used the same terminology throughout this paper. In this section, we shall examine first which papers have dealt with the VRP (in an incomplete manner, as this problem has been exten-

sively researched), and then examine what research has been done on various types of mobile health care problems.

## 2.1 VRP

The Vehicle Routing Problem calls for the determination of an optimal set of routes to be performed by a fleet of vehicles to serve a given set of customers. It is one of the most important combinatorial optimisation problems.

The first paper to deal with the VRP specifically was written by Dantzig and Ramser in 1959 [5]. In their paper, the authors described a real-world application and proposed the first mathematical model and algorithmic solution to the problem. In 1964, Clarke and Wright [4] proposed a greedy heuristic that improved on the approach put forward by Dantzig and Ramser. In the following years, many papers have been written describing mathematical models, and both exact and heuristic algorithms for solving either optimally or approximately many variants of the VRP. We shall however refrain from examining all the research that has beeen done on the VRP, but rather focus on those papers that deal with heuristic or meta-heuristic solution methods, given that this paper itself deals with a meta-heuristic solution method. The papers that deal with these methods can be divided into "classical heuristics" mostly described before about 1995, and metaheuristic algorithms afterwards.

### Classical Heuristics

As stated above, the first heuristic proposed for the VRP was the savings algorithm described in 1964 by Clarke and Wright [4]. This algorithm starts with as many routes as there are customers, and for each pair of routes, computes the time (or distance) that would be saved by merging them. Then the routes

are merged together starting with the greatest saving, while respecting the constraints.

In 1974, Gillett and Miller [9] introduced the sweep algorithm. This algorithm relies on the spatial layout of the customers. Using the depot as the origin it sorts the customers by their polar coordinates, and adds them in increasing order to a route, changing routes when the vehicle's capacity is exceeded. This heuristic is very fast, and usually yields good results, and has therefore been used extensively in the literature.

A natural extension to the sweep algorithm is the petal method, where one generate several routes, called *petals*, and making a selection by solving the set partitioning problem. This method was first proposed by Balinski and Quandt in 1964 [1].

Another algorithm introduced by Fisher and Jaikumar [8] only a few years later, in 1981 is the cluster first, route second algorithm, where routes are constructed by grouping customers in clusters of which the total demand does not exceed the capacity of the vehicle and solving one TSP per cluster. Because each cluster is quite small, the TSP solution is easy to find, and therefore the algorithm is fast.

Several inter-route improvement methods have been proposed over the years. In 1965, Lin [14] proposed the $\lambda$-opt mechanism where $\lambda$ edges are removed from a tour and the $\lambda$ remaining segments are reconnected in all possible ways. A profitable reconnection (the first or the best) is identified and implemented. When there are no more profitable reconnections, a local optimum is reached and the procedure stops. Several modifications of this approach have been published, most notably Lin and Kernighan in 1973 [15] who modify $\lambda$ dynamically throughout the search. Thompson and Psaraftis in 1993 [20], Van Breedam in 1994 [21] and Kinderwater and Savelsbergh [12] in 1997 proposed multi-route

edge exchange schemes for the VRP which were significantly reused afterwards. In 1996 Xu and Kelly [23] amongst others, introduced Ejection Chains.

**Metaheuristics**

These types of solution strategies are by far the most promising, on the whole. Overall results are better than classical heuristic methods. There are three different types of metaheuristics which apply to the VRP: local search based methods, population search methods, and learning mechanisms. We shall restrict our survey to local search based methods, for that is what our solution strategy is based on.

*Local Search* There are several local search based methods. In 1989, Glover [10] introduced Tabu Search, which is a form of local search where, to avoid cycling, and therefore local minima, once a move has been made within the neighbourhood, the correspond inverse move is forbidden, at least for a time. This is because this method is based upon the idea that within a local search neighbourhood, if a given move yields a good solution, then the inverse move is likely to yield a bad one. Therefore the algorithm, upon reaching a better solution than the previous one, stores in memory the move that allowed it to reach the better solution, and forbids itself for a specified number of moves from performing the opposite move. In 1989, Willard [22] was one of the first to apply this method to the VRP. This method was improved several times, and remains one of the most applied metaheuristics to the VRP.

A metaheuristic that is derived from the Tabu Search is the Taburoute, described by Gendreau, Hertz and Laporte in 1994. In this method, vertices are removed from a route and put into another one (all the possible move of this type constitute the neighbourhood). It is then "forbidden" from returning to

that route for $\theta$ iterations, where $\theta$ is contained in $[5, 10]$. Taburoute also uses a *diversification* strategy, penalizing vertices that have been frequently move in order to increase the probability of considering slow moving vertices.

Another local search based metaheuristic is Simulated Annealing, which models the way liquids freeze in the process of annealing, going from a chaotic high-energy state to a more ordered solid state. The process starts at a temperature $T$. Variations in energy are observed, ie the local search moves from one solution to another. If the new energy is higher (the solution is worse), it is accepted with probability $e^{-\frac{dE}{T}}$. This allows the system to escape local minima. After a number of iterations, enough to give good sampling statistics, the temperature is decremented and the process starts anew.

The Adaptive Large Neighbourhood Search, or ALNS, is another metaheuristic that is based on local search. However it can be instegrated with the other methods descirbed above. The principle is the following: there exist many heuristics which do a good job of inserting requests into a VRP solution. However they are all bound by their susceptibility to local minima. The ALNS uses these heuristics as the neighbourhood itself, alternately removing and reinserting requests into the solution using destroy and repair heuristics. As the method proceeds, the heuristics are weighted according to how well they perform, and at each iteration a destroy/repair pair is chosen according to this adaptive roulette wheel.

Most notably, Pisinger and Ropke [16] used an Adaptive LNS to solve five different variants of the vehicle routing problem: the vehicle routing problem with time windows (VRPTW), the capacitated vehicle routing problem (CVRP), the multi-depot vehicle routing problem (MDVRP), the site-dependent vehicle routing problem (SDVRP) and the open vehicle routing problem (OVRP), all of

which present many similarities with our problem. They also recommended [17] the LNS specifically for Vehicle Routing Problems (VRP).

### 2.2 Mobile Nurses

The problem of mobile nurses has been studied significantly. We were able to uncover several papers dealing with this type of problem. Begur et al [2] described a system for scheduling and routing of home health care nurses which integrated GIS software with scheduling heuristics and databases, which through better scheduling, improves routing, and the balance of work among nurses. Bertels and Fahle [3] described in 2005 a system (Parpap) which through a combination of linear programming, constraint programming, and meta-heuristics was able to efficiently find solutions to home health care problems. Eveborn et al [7] were also able to significantly improve the level of care given to the elderly in Sweden using the Laps Care system they developed [6], which uses a repeated matching heuristic.

The research that comes closest to our problem, albeit with no treatment of time windows, is the 2009 paper by Groër et al [11]. In this paper, they consider a problem from the small package shipping industry, which is a variant of the classical capacitated VRP, which they call the Consistent VRP, or ConVRP, because of the constraint that the same drivers must visit the same customers on each day that the customers need service. The algorithm they use to solve the problem is based on the Record-to-Record algorithm introduced by Li et al [13] in 2005 to solve large-scale VRPs.

## 3 Contribution

To find results for the ConVRP, we use an LNS-based heuristic. First we construct, by means of a heuristic, an initial solution, as good as possible. Then

we enter the improvement phase, where we use a metaheuristic based on local search. For the neighbourhood, we chose to follow Shaw [19] and use a Destroy and Repair neighbourhood, for several reasons. First, this method is very powerful, and yet simple to apply to our problem. Also it allows us to easily implement several different types of destroy heuristics, and several different types of repair heuristics, which allows us to test different methods. Also, given the nature of the problem, which is quite constrained, our hope is that the removal and reinsertion of relatively large numbers of clients at once would lead to great jumps between solutions being possible.

### 3.1 Initialisation Phase

For the initialisation phase of the algorithm, the aim is to construct an initial selection of routes. These routes need to be if possible complete, ie each request is served by a nurse, and as good as possible, all the while being calculated as fast as possible. To this end we use a construction heuristic. In fact, any one of the insertion heuristics presented below can be used, but we found that the best compromise between speed and quality came from the regret insertion heuristic. As we also used this procedure for inserting requests as a part of the Destroy and Repair neighbourhood, we will not go into detail here. Sufficie it to say that we create one route per nurse per day, and fill these routes with the requests using the regret insertion procedure. This leaves us with a valid solution, which we could then improve on.

### 3.2 Improvement Phase

The improvement phase is the most important part of the solution strategy. This is where the initial solution is improved upon by means of a local search based algorithm. For this, we chose the simulated annealing framework. The reason

for this choice is that our neighbourhood is very large scale, and quite random. Therefore we felt that a system that allows many iterations to be made, is not too prone to becoming trapped in a local minimum, and works well with randomised neighbourhoods. Simulated Annealing presented all these characteristics, as well as being very well represented in the literature.

The inspiration for Simulated Annealing comes from metallurgy, where annealing is the process of heating a metal and cooling it gently, giving time for the atoms to form into larger crystals, thereby reducing defects. In the same way, each step of the simulated annealing method chooses a random neighbour for the current solution (or energy state). This choice is made in such a way as to leave a chance that a higher energy state (or worse solution) can be chosen, allowing the method to escape local minima. This probability becomes smaller as the temperature decreases as time goes on, and the method comes to resemble a simple greedy local search. The algorithm is stopped when the temperature becomes low enough. The temperature is decreased after a certain amount of iterations, and the process starts anew. Simulated Annealing is considered a Monte Carlo method because of it's reliance on repeated random sampling.

In Algorithm 1 on the next page we present the Simulated Annealing method.

**Neighbourhood**

As we stated above, we have followed Shaw in using a Destroy and Repair neighbourhood. More specifically, in the Simulated Annealing framework, at each iteration, we randomly choose an algorithm to delete requests from the solution, and an insertion algorithm to reinsert the deleted requests into the solution. A request pool is kept for this purpose, and also to keep track of any requests

**Algorithm 1**: Simulated Annealing

```
Input: initial solution S_init, α_init, T_init, T_final, plateau length p
Output: final solution S_final
T ←T_init;
S_final ←S_init;
S ←S_init;
while T > T_final do
    count ←0;
    while count < p do
        S_next ←N(S);
        if S_next < S then
            S ←S_next;
        else
            r ←Rand(0,1);
            if e^(−(S_next − S)/T) < r then
                S ←S_next;
        if S < S_final then
            S_final ←S;
        count ←count + 1;
return S_final
```

that might not have been inserted for some reason. In the next section, we shall discuss which algorithms we implemented, and why.

The real power of this method lies not only in the insertion heuristics that are chosen, but also, and maybe even more so in the procedures that are used to partially destroy solutions. Shaw [19] goes into more detail in his paper. Mainly the idea is that it is useful to remove requests that are related in some way. He gives as possible examples requests that are in the ssame route, and requests taht are geographically close to one another. In our case we have prefered not to remove requests in the same route, because we are aiming for consistency, and removing entire "route-loads" of requests would not advance us towards our goal, nor would it go against any possible deficiencies in our solution. Shaw recommends it to reduce the number of routes, whereas we have no desire to

do such a thing. In fact we would prefer there to be more routes, and therefore more equal shifts for the nurses.

We have however placed more focus on the spatial aspect of the relatednesss issue, as we implemented a destroy heuristic that removes requests that are inserted in such a way as to cause a "spike", or undue added distance in the route.

### 3.3 Algorithms

Here we present the different low-level heuristics that were used to construct the neighbourhood that was needed for the Simulated Annealing.

#### Inserters

*Greedy Insertion* The first insertion algorithm we propose, the greedy algorithm, is also one of the simplest ones. It is presented in Algorithm 2 on the following page. The basic idea is that of finding the best insertion position for a given request and inserting it at that position. The algorithm, for each request, searches through all the routes, saving the current insertion cost if it is better than the previously saved one. When it has finished going through all the routes it inserts the request at the saved position in the corresponding route. This algorithm, while fast, has several drawbacks. First of all it is myopic, because it does not consider the other requests when finding the best insertion route and position. Also because the algorithm takes the requests in the order they come in, that order changes the result of the algorithm. We kept it nevertheless because although myopic, it does tend towards better solutions when included as part of an SA.

*Random Insertion* The second algorithm we present (see Algorithm 3 on page 17), the random algorithm, is in fact only random when it comes to the choice of

**Algorithm 2**: Greedy Insertion

```
Input: List of routes, List of requests
Output: List of routes
foreach request re do
    found ←false;
    bestCost ←M;
    foreach route ro do
        foreach position p do
            if insertion is possible then
                if cost of insertion ¡ bestCost then
                    bestCost ←cost of insertion;
                    bestPosition ←p;
                    bestRoute ←ro;
                    found ←true;
    if found then
        insert re into bestRoute at bestPosition;
    add re to uninserted vector;
```

which route it inserts each request into. In other words, it chooses a random route. It then checks the insertion cost for each position in the route, saving the best one, and inserts the request at that position. This algorithm, although it presents the obvious drawback of possibly choosing the worst route to insert into, has been included because it allows the LNS to "mix up" the results a little, changing the nature of the neighbourhood, and therefore potentially allowing better results to be found. However it is not as bad as an algorithm which would chose a random position in a random route, and so we felt it presented a good compromise between optimisation and randomness.

*Regret Insertion* The third insertion algorithm, which we also use to initialise our routes, was applied to the VRPTW in 1993 by Potvin and Rousseau [18], although it dates back to the seventies. It is called the Regret insertion algorithm, because it is based on the idea that instead of simply inserting at the best possible

**Algorithm 3**: Random Insertion

```
Input: List of routes, List of requests
Output: List of routes
foreach request re do
    make list of possible routes;
    found ←false;
    while !found do
        route ←select random route from possible routes;
        bestCost ←M;
        foreach position p in route do
            if insertion of re is possible in rooute at p then
                if cost of insertion ¡ bestCost then
                    bestCost ←cost of insertion;
                    bestPosition ←p;
                    found ←true;
        if !found then
            remove route from possible routes;
            if vector is empty then
                add r to rejected requests;
                found ←true;
            else
                choose new route from eligible vector;
        else
            insert re into route at bestPosition;
```

position in the best possible route, one can find out for each request, how much one would "regret" not inserting at that position. The algorithm (presented in simplified form in Algorithm 4 on the following page) finds for each request the best insertion position in each route. This best insertion position is the one that minimises the related cost, considered as an increment of the value of the objective function. If there is no possible insertion in a route, the insertion cost is set to an arbitrarily high value. Next the regret value is calculated, which for a request is the difference in insertion costs between the best position and the

second best position. This, because of the impossible insertions being set to a high cost, always gives us a positive number. The request with the highest regret is inserted at it's best position in the corresponding route.

In our implementation, we added a caching mechanism to this calculation. If after a request has been inserted, there remain requests to deal with, the insertion values for the remaining requests only change for the route that has just had a request inserted. Therefore it is possible to cache these values, which leads to a considerable speed up, especially when the routes are already long. This is particularily useful in the improvement phase of the algorithm, when only some of the requests are removed, leaving the routes partially complete.

**Algorithm 4**: Regret Insertion

```
Input: List of routes, List of requests
Output: List of routes
while requests remain do
    foreach request re do
        make list of eligible routes;
        foreach eligible route ro do
            max ←M;
            foreach position p do
                if cost c of insertion in ro at p is lower than max then
                    maxpos ←p;
                    max ←c;
        regret for request is the second best cost minus the best cost
    insert request with highest regret into best route at best position;
```

### Deleters

Following Shaw et al [19], we have implemented several deletion heuristics that are based on *relatedeness*. All of these methods take as a parameter the percent-

age of requests to remove from the solution, and all of them remove requests until that percentage is reached.

*Consistency Deletion Operator* This deletion operator (see Algorithm 5) first finds the client wich is the least consistent, ie the one which is served by the greatest number of different nurses. Having found this client, it removes from the routes all of the requests belonging to it. If at this point, the solution size has been reduced by the supplied percentage, the algorithm returns the list of removed requests, otherwise it selects the next least consistent client and starts anew. This algorithm is obviously the most relevant to the consistency constraint.

**Algorithm 5**: Consistency Deletion Operator

```
Input: Solution s, Percentage p
Output: List of deleted requests
dl ←empty list;
while p not reached do
    find least consistent client c;
    get l list of requests belonging to c;
    foreach request re in l do
        add re to deleted list dl;
        remove re from s;
return dl;
```

*Random Deletion Operator* The algorithm presented in Algorithm 6 on the following page is mostly the same as the Consistency Deletion Operator (see Algorithm 5). However, instead of selecting the requests to remove by choosing the least consistent client, it chooses a random client. This is in the hope that results will be more varied, and therefore we would have a better chance of finding interesting results, even though there is the risk of removing a client whose requests are all optimally inserted.

**Algorithm 6**: Random Deletion Operator

```
Input: Solution s, Percentage p
Output: List of deleted requests
dl ←empty list;
while p not reached do
    select random client c;
    get l list of requests belonging to c;
    foreach request re in l do
        add re to dl;
        remove re from s;
return dl;
```

*Distance Deletion Operator* This last deletion operator does not consider consistency, but rather routing. The idea behind it is that for a given solution, it is advantageous to remove those requests which have a disproportionate effect on the cost of their route. So the algorithm removes the requests with the largest average distance from their neighbours. These requests are the ones that form "spikes" in the routes in many cases.

Consider a request $x$, which is preceded by $i$ and followed by $j$ in the route. The average distance from the neighbours of $x$ is

$$\frac{D(i,x) + D(x,j)}{2}$$

where $D(i,j)$ is the distance from $i$ to $j$. This operator removes the specified percentage of the most distant requests. This algorithm is presented in Algorithm 7 on the facing page.

### 3.4 Implementation

We have implemented the above algorithms in C++. This language presents several specific advantages which we found to be most useful in our case. First, it is compiled, and therefore fast. Given that our simulated annealing framework

**Algorithm 7**: Distance Deletion Operator

```
Input: Solution s, Percentage p
Output: List of deleted requests
dl ←empty list;
sl ←list sorted by distance of requests from their neighbours;
foreach request re in s do
    insert re into sl in a sorted manner;
    if size of sl greater than p then
        truncate sl;
foreach request re in sl do
    add re to dl;
    remove re from s;
return dl;
```

runs for a fixed number of iterations, the faster the language, the faster the results are calculated. Also it allows the programmer access to object oriented programming, a real boon given the metaheuristic and modular nature of the method. We were able to write the framework without regard to the implementation of the routes, or the operators. Also we were able to leverage Virtual Inheritance to great effect, both on the data importing side of the software, and on the core functionality, like the deletion operators. As it stands, we can quickly implement a new deletion or insertion operator without changing any code in the SA framework.

## 4 Computational Results

### 4.1 Parameters

#### Simulated Annealing

The Simulated Annealing framework accepts several different parameters. The first is the initial temperature, which effectively sets how long the method will run. Secondly, there is the end temperature $T_{final}$, when the metaheuristic ex-

its. Next is the length of the plateau, which sets how long each energy state is examined. It is important to not set this too short, so that a statistically significant number of states can be examined. Finally there is the rate at which the temperature decreases. This is achieved by multiplying T by $\alpha$, a number between 0 and 1.

In our tests, we initially set the temperature to 100. However over the course of our tests, we found that it was not necessary to start so high, and we set it to 60 instead. This afforded us with a decent exploration of the solution space, while keeping the resolution time reasonably short. The final temperature was set to 5 for the same reasons.

The plateau length was set to 150. Initially we started at 100, but solutions were not improved as much as expected, so we set it to 150, which delivers better results.

| $T$ | $T_{final}$ | $p$ | $\alpha$ | Time (seconds) | Length | Consistency |
|---|---|---|---|---|---|---|
| 100 | 1 | 100 | 0.9 | 4.5 | 2595.1 | 6000 |
| 100 | 5 | 100 | 0.9 | 3.2 | 2900.37 | 6000 |
| 60 | 1 | 100 | 0.9 | 2.8 | 2678.78 | 9000 |
| 60 | 5 | 100 | 0.9 | 2.2 | 2827.97 | 7000 |
| 100 | 1 | 150 | 0.9 | 3.8 | 2633.07 | 7000 |
| 100 | 5 | 150 | 0.9 | 1.7 | 2668.02 | 7000 |
| 60 | 1 | 150 | 0.9 | 3.9 | 2752.75 | 9000 |
| 60 | 5 | 150 | 0.9 | 1.4 | 2910.05 | 9000 |
| 100 | 1 | 100 | 0.99 | 37.9 | 2777.13 | 5000 |
| 100 | 5 | 100 | 0.99 | 26 | 2599.18 | 5000 |
| 60 | 1 | 100 | 0.99 | 38.7 | 2698.5 | 5000 |
| 60 | 5 | 100 | 0.99 | 22.3 | 3011.19 | 7000 |
| 100 | 1 | 150 | 0.99 | 38.3 | 2628.68 | 7000 |
| 100 | 5 | 150 | 0.99 | 27.3 | 2831.47 | 6000 |
| 60 | 1 | 150 | 0.99 | 34 | 2542.05 | 8000 |
| 60 | 5 | 150 | 0.99 | 24.5 | 2603.89 | 7000 |

Fig. 1: Parameter settings comparison

Figure 1 on the preceding page shows the differences between the various settings. For each parameter we have chosen two possible values, and run the algorithm on the same problem, which has 12 clients, 66 requests, and 3 nurses. The first difference that catches the eye is between the set of those runs with $\alpha$ set to 0.9 and those with $\alpha$ set to 0.99. Alhtough the results are generally better in the latter case, the cost in time is prohibitive, as it multiplies the run duration by an order of magnitude.

The second interesting figure is the difference between the runs where $T_{final} = 1$ and those where $T_{final} = 5$. There is not much difference between them in the cases where $\alpha = 0.9$, except for the running time, which is larger when $T_{final}$ is smaller. This makes sense because at the lower end of the temperature scale, there is a very small probability that a worse solution would be accepted in lieu of the current one. Therefore prolonging the run at low temperatures yields little benefit with regards to the time spent.

The most debatable parameters in this instance are the $p$ and $T$ values. It seems that the solutions are slightly better when $p$ is larger. this makes sense because the algorithm has more time to explore the neighbouring solutions at that state of energy, and especially while the temperature is high, can escape a potential local minimum. In the same way a high $T$ leads to slightly better solution, but the effort is not always worth the payoff.

In the end we selected $T = 60$, and $p = 150$, which seemed to give a good compromise between speed and result quality.

### Destroy and Repair Neighbourhood

The last parameter is the number of requests that should be removed and reinserted at each step. In *A general heuristic for vehicle routing problems* [16],

Pisinger and Ropke recommended between 10 and 40 percent of the solution size. We decided to use a conservative interpretation of this recommmendation, and set the percentage to 10. Of course, given that the heuristics only stop removing requests when they have removed more than the given percentage, the actual number is often exceeded, especially on smaller instances, which is in line with Ropke and Pisingers method, where they recommend the number should be limited on larger instances.

### 4.2 Comparison to Groër et al

**Results**

We wrote a parser for the data proposed by Groër et al. There are two sets. One set are small problems with 10 or 12 customers, with a schedule of 3 days, randomly generated by Groër et al. The other are randomly generated from the well-known Christofides instances where each customer has a probability of p=0.7 of being visited on each of the 5 days of the schedule. All distances in both sets are euclidian. There are no time windows, and there is at most one demand per customer per day. In order to compare our results with those of Groër et al, we set the time windows in each case to the whole day, and disregarded the demand values. We also set the number of nurses to the same number as the number of vehicles in the corresponding Groër instance.

In Figure 2 on the facing page we present the results of the comparison on some of the larger instances from Christofides et al. The main difference that we can see is that where Groër et al have included a consistency constraint, we have made consistency part of the objective function. This means that the routing values in our case are better, but some of the requests for certain clients are served by a different nurse. Each time this happens, our objective function is increased by 1000, which leads to the observed consistency figures. all of our

results in this table are averaged over 5 runs, however, which is why the consistency values are not necessarily a multiple of 1000. This is most notable in cases where there are a lot of requests for not many clients, for example problem 9.

But on the whole, we can see that our method finds results which are as good as those of Groër et al. Also, and especially, our consistency values, while not optimal in every case, are usually very good.

| Problem | Customers | Requests | Fleet size | Groër travel time | Length | Consistency |
|---|---|---|---|---|---|---|
| 1 | 50 | 164 | 5 | 2282.14 | 2147.25 | 1000 |
| 2 | 75 | 268 | 11 | 3872.86 | 2949.56 | 1000 |
| 3 | 100 | 337 | 7 | 3628.22 | 3020.53 | 1000 |
| 4 | 150 | 544 | 12 | 4952.91 | 3903.24 | 1750 |
| 6 | 50 | 173 | 5 | 4084.24 | 2347.6 | 1800 |
| 7 | 75 | 276 | 12 | 7126.07 | 3310.33 | 20000 |
| 9 | 50 | 534 | 14 | 11033.54 | 5539.74 | 63333.3 |
| 11 | 120 | 425 | 7 | 4753.89 | 3612.21 | 1000 |
| 12 | 100 | 325 | 10 | 3861.35 | 3120.27 | 1000 |

Fig. 2: Comparison to Groër et al

**Graphical interpretation**

In Figure 3 on page 27 we present some graphical views of the results of the solution to the smallest of the problems proposed by Groër et al.. One can see that the routes are not completely optimal when it comes to routing. Given that in these instances there are no time windows, it seems easy to optimise the routes further so that there are no more loops. However, this situation is explained by the neighbourhood operators we have implemented. Out of three deletion operators, only one is based on routing, and even then it removes requests that are far away from their neighbours, with no regard to order or looping. the other

deletion operators remove requests by client, and have no concept of routing whatsoever. When it comes to the insertion operators, they are all based on the objective function, whcih is heavily weighted towards consistency. These two factors taken together lead to a system which would naturally find it hard to detect and remove loops.

The second issue that catches the eye, especially in 3(a), is the fact that the routes are unequal when it comes to length. This means that some nurses will have longer shifts than others. While this was never part of the original problem, it would not be hard to solve simply by creating a deletion operator which removed requests from the longest routes, hoping that they would then be reassigned to a different nurse. Of course to keep consistency, the requests would have to be removed by client also, but this should not be hard to do.

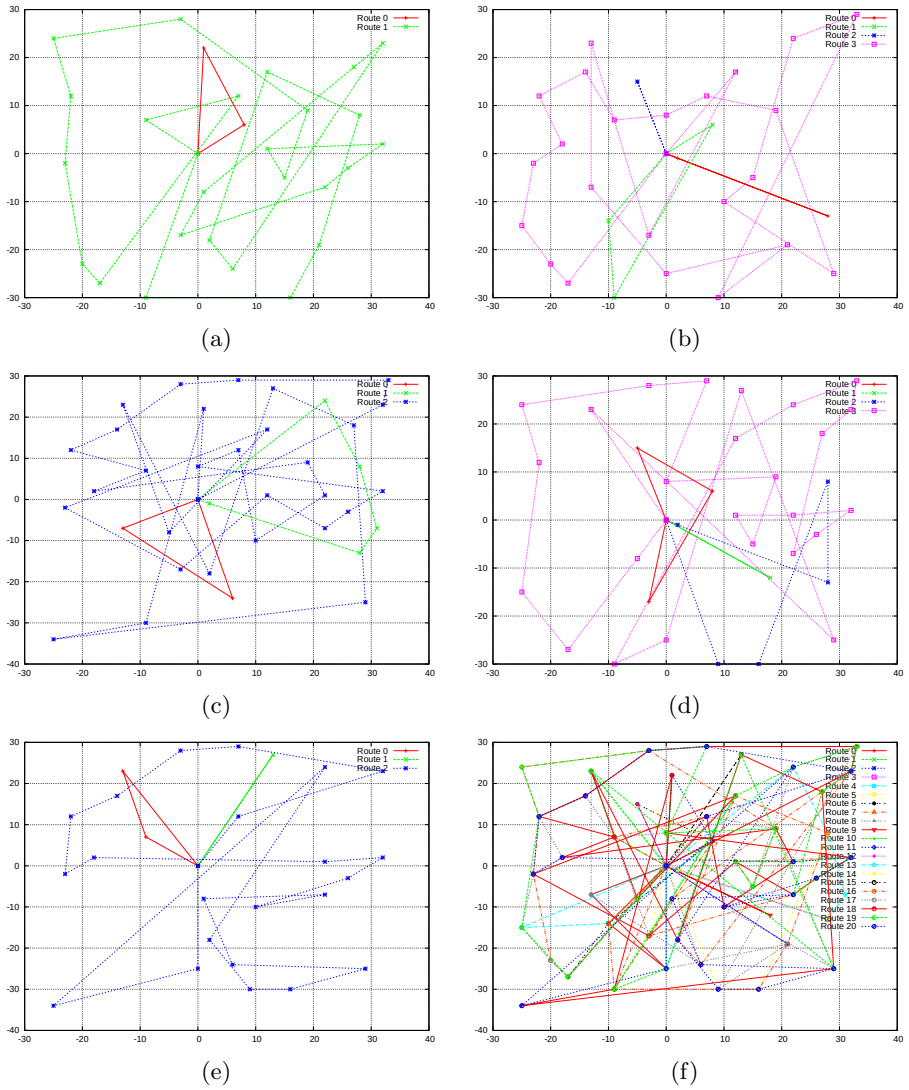


Fig. 3: Routing broken down by day: (a) Day 1, (b) Day 2, (c) Day 3, (d) Day 4, (e) Day 5, (f) All days

## 5 Conclusions

The mobile health care industry (for want of a better word), as may be guessed, attaches great importance to the comfort of it's patients. This can in part be accomplished by allowing a given nurse to build a working relationship with his or her patients. To do this, it is necessary for patients to see the same nurse as often as possible over the course of their various treatments.

In this paper we developed a method for generating consistent mobile nurse routes. On the generated instances, the Simulated Annealing framework we presented, based on LNS, was able to perform very well.

However, there are several ways in which this method could be improved further. First of all the destroy and repair heuristics could, instead of being selected randomly, be selected using a roulette wheel, with more weight given to the less random or greedy algorithms. This would have the advantage of prioritising better heuristics. another way to do this would be to use the adaptive method described in the literature, where the roulette whelle is initially composed of equal probabilities, but during the search gets biased towards those pairs of heuristics that perofrm better than other pairs.

Using the flexibility of the proposed method, it would be relatively simple to include other heuristics, such as a deletion operator which tries to remove requests that are involved in loops within the route. Another possible deletion heuristic is one that removes requests that belong to the same route. Overall, any deletion operator that removes request that are related, for any definition of related, is in all probability a welcome addition to the method.

With regards to the method itself, we have presented an implementation of a very well-respected metaheuristic framework applied to a new class of problem. Our implementation has stood up to the test of the data instances generated by Groër et al, and therefore we feel that the results are satisfactory.

# Bibliography


[1] Balinski, M. and Quandt, R. (1964). On an integer program for a delivery problem. *Operations Research*, 12(2):300–304.

[2] Begur, S. V., Miller, D. M., and Weaver, J. R. (1997). An integrated spatial dss for scheduling and routing home-health-care nurses. *INTERFACES*, 27(4):35–48.

[3] Bertels, S. and Fahle, T. (2006). A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. *Computers and Operations Research*, 33(10):2866–2890.

[4] Clarke, G. and Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

[5] Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

[6] Eveborn, P., Flisberg, P., and Rönnqvist, M. (2006). Laps. *European Journal of Operational Research*, 171(3):962–976.

[7] Eveborn, P., Rönnqvist, M., Einarsdóttir, H., Eklund, M., Lidén, K., and Almroth, M. (2009). Operations research improves quality and efficiency in home care. *Interfaces*, 39(1):18–34.

[8] Fisher, M. and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124.

[9] Gillett, B. E. and Miller, L. R. (1974). A Heuristic Algorithm for the Vehicle-Dispatch Problem. *OPERATIONS RESEARCH*, 22(2):340–349.

[10] Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166.

[11] Groër, C., Golden, B. L., and Wasil, E. A. (2009). The consistent vehicle routing problem. *Manufacturing & Service Operations Management*, 11(4):630–643.

[12] Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. *Local Search in Combinatorial Optimization*, pages 337–360.

[13] Li, F., Golden, B., and Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32(5):1165–1179.

[14] Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

[15] Lin, S. and Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

[16] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & OR*, 34(8):2403–2435.

[17] Pisinger, D. and Ropke, S. (2009). *Handbook of Metaheuristics, 2nd edition*. none. (to appear).

[18] Potvin, J.-Y. and Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331 – 340.

[19] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming—CP98*, pages 417–431.

[20] Thompson, P. and Psaraftis, H. (1993). Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. *Operations Research*, 41(5):935–946.

[21] Van Breedam, A. (1994). *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints.* PhD thesis, University of Antwerp.

[22] Willard, A. (1989). Vehicle routing using r-optimal tabu search. Master's thesis, The Management School, Imperial College, London.

[23] Xu, J. and Kelly, J. (1996). A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30(4):379.