**UNIVERSITATEA ”ALEXANDRU IOAN CUZA”, IAȘI**

**FACULTATEA DE INFORMATICĂ**

LUCRARE DE LICENȚĂ

# Vehicle routing and scheduling for regular mobile healthcare services

propusă de

**Cosmin-Ștefan Pascaru**

**Sesiunea:** Iulie 2018

Coordonator științific

**Conf. Dr. Mihaela Elena Breabăn**

**UNIVERSITATEA ”ALEXANDRU IOAN CUZA”, IAȘI**

**FACULTATEA DE INFORMATICĂ**

LUCRARE DE LICENȚĂ

# Vehicle routing and scheduling for regular mobile healthcare services

propusă de

**Cosmin-Ștefan Pascaru**

**Sesiunea:** Iulie 2018

Coordonator științific

**Conf. Dr. Mihaela Elena Breabăn**

Avizat,

Îndrumător Lucrare de Licență

Titlul, Nume și prenume: Conf. Dr. Breaban Mihaela

Data ......................................

Semnătura ......................................

**DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemntatul Pascaru Cosmin-Ștefan, domiciliul în Str. Friederick Nr. 14, orașul Iași, județul Iași, România, născut la data de 26 decembrie 1996, identificat prin CNP 1961226226711, absolvent al Universității ”Alexandru Ioan Cuza” din Iași, Facultatea de informatică, specializarea informatică, promoția 2015-2018, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul *Vehicle routing and scheduling for regular mobile healthcare services*, elaborată sub îndrumarea d-nei Conf. Dr. Mihaela Elena Breabăn, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop. Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, ............................. Semnătură student .............................

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul *Vehicle routing and scheduling for regular mobile healthcare services*, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea ”Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, Data: ................................

Absolvent: Cosmin-Ștefan Pascaru

Semnatura: ...................................

# Contents

# Introduction and Motivation

In 2016, Romania had an *infant mortality rate* (IMR) of 7.0. This value represents the number of deaths of children under one year of age per one thousand births. This is the highest out of all member states of the European Union. This indicator is obviously extremely representative of how well developed a country is. The health-care system is one of the main entities that are responsible for reducing it as much as possible. One essential condition is to increase the number of ultrasounds and prenatal examinations that pregnant women take, especially in rural areas. Such tests can reveal a large number of health issues that could be easily treated, if found in time.

This problem is what caused the need for a project that proposes the acquisition of a number of vans and equipping them with the necessary medical devices (most importantly, an ultrasound machine), a medic, a driver, and scheduling them on daily routes.

The project was initiated only for the Iași county. If successful, it can be applied in other counties as well. The destinations of interest are the communes around the county capital, as they do not have their own hospitals. Other main towns close to the county capital are excluded since they have their own hospitals, and as such, much better access to medical services. There are $93$ such townships and the driving distance between any of them is under *three* hours. Each daily route of a van contains a list of communes to visit in that specific order, and perform a number of examinations in each one, according to the planning for that specific day.

The project is part of a nationwide initiative to improve general health-care in disadvantaged areas in Romania. Considering the relatively poor infrastructure of rural areas, lack of hospitals and specialized personnel, this depends on using *mobile* medical units. Because of this, it was necessary to modify the legislation to allow them and include proper methodological norms. A long time after the first proposal, this was finally approved by the Ministry of Health of Romania in May 2018 [1].

According to data provided by Romanian National Statistics Institute (INS) [2], in 2016, in

[1]http://www.cdep.ro/pls/legis/legis_pck.htp_act?ida=151171
[2] http://www.insse.ro/cms/en

Iași county, $9.88\%$ live births occurred without any prenatal testing, and another $12.43\%$ had delayed their first examination to the second or even third trimesters. This excludes towns that are located less than $10$ kilometers away from the county capital.

The start and end points of any daily route are in the county capital. All the driving distances between any pair of destinations were determined by calls to the Google Maps API [3] beforehand and saved as constant data for the scheduling algorithm. This results in a reasonable approximation, as real-time traffic delays are impossible to be considered at the planning stage, which is done before any departure.

The initial motivation for this work is the budget estimation for this previously described project. The first concern is the initial cost that depends on the price and amount of required resources. Medical equipment is very expensive, so the most relevant metric is the number of required vans with proper equipment. This depends on the quantity of provided services, i.e. number of medical investigations, township configuration and route distances. The first phase was to gather statistics about how many patients lack the provided services in each location, based on previous years data, and then give a rough approximation of best, average, and worst case costs.

[3] https://cloud.google.com/maps-platform/

# Chapter 1

# The Tackled Optimization Problem

## 1.1 Problem Modeling

### 1.1.1 Prior design decisions

The initial goal of the project was to determine the minimum number of vans that would ensure all the examinations needed for all the pregnancies that are estimated to occur in the following year. In addition, the developed algorithm is able to compute the routes themselves and to schedule of vans by days. The approximation of the number of pregnancies is the first step in solving the problem.

An important decision that was made at the beginning of the project was to work with communes instead of villages. The decision was influenced by the fact that every village is close enough that a person can reach the center of the commune it is annexed to. Furthermore, the number of villages in Iași county is $331$ and the number of communes is $93$. Working with a lower number of locations provide more flexibility in the design of the chosen solution that will be presented in the following chapters.

### 1.1.2 Data description and preprocessing

INS provided detailed data about the pregnancies examination between years 2000 and 2016. The provided data contained, for each commune, the total number of live births, out of which how many had no prenatal examination, and that had their first examination in first, second and third trimesters of pregnancy.

It was decided only the data between $2014$ and $2016$ to be used and it was summed up the total number of births, without taking into account the examination in any of the trimesters, and divided by *three*. Since the recommended number of examinations during a pregnancy is *seven*, for each commune the number total number of births was multiplied by *seven*. The scheduling should be provided for each month so the obtained number is divided by *twelve* and rounded to the higher integer.

The formula described above is built under the assumption that all the all months follow the same schedule, only patients change and that births are evenly distributed by months. It will be considered that every month contains in average *twenty-one* working days and a day has maximum of *ten* working hours.

### 1.1.3 Result description

The algorithm is able to compute a route that contains an ordered list of communes and, for each commune, the associated number of examinations. A route should be completed in one day, which in fact means *ten* working hours. In order to compute the duration of a route, the driving times should be summed up and 30 minutes multiplied by the total number of visits on a route should be added to the sum.

The problem challenge is to find the minimum number of routes that cover all the examinations. If this number was minimized than It automatically would be minimized the number of vans since it equals the number of routes divided by 21, rounded to the higher integer. Vans are of the same kind, so there is no preference regarding which van should handle one route. The vans are just assigned to a different route each day.

## 1.2 Problem Definition

This section presents a formal definition of the problem and how it differs from some similar classical problems presented in section 1.3.

### 1.2.1 Parameters

A series of constant parameters have values assigned based on practical considerations. In the algorithm that was developed these parameters are stored in *a configuration file* and they are:

- the duration of the examination (30 minutes);
- the number of working hours (10 hours or 600 minutes);
- the number of working days in a month (21 days).

### 1.2.2 Input

The problem input if formed out of:

- communes configurations;
- distances between communes;
- number of examinations to be performed in each commune.

The distances between each pair of communes and each commune and the capital were pre-computed and stored in a $(N+1)\times(N+1)$ matrix, note that $N = 93$ is the number of communes in Iași county. The distance matrix contains only integer values, representing driving distance expressed in minutes. Instead of the commune name, a numeric id was used because it is easier for the core algorithm to handle numeric values. The mapping between names and ids was kept in order to provide user-friendly output.

In the following sections the below notations will be used:

- $E_i$ represents number of examinations necessary in township $i$
- $D_{i,j}$ travel time distance between locations $i$ and $j$, it will be considered that $0$ is the capital

### 1.2.3 Output

The output consists of a list that contains the scheduled tours, which can cover all the needed examinations. A tour is made of the communes that are visited and the number of examinations performed in each one. In the following sections the below notations will be used:

- $T$ represents the complete list of tours (can also be viewed as a set, as the order in which tours are made is not important);
- $T_i$ represents the $i^{th}$ tour, $1 \leq i \leq M$, where $M$ denotes the number of tours;
- $T_{i,j}$ denotes the $j^{th}$ commune id of tour $i$, where $1 \leq j \leq |T_i|$, where $|T_i|$ represents the tour length
- $S_{i,j}$ represents the number of examinations scheduled at $T_{i,j}$ in tour $i$, where $1 \leq j \leq |T_i|$

One solution is valid if it satisfies all required examinations and if every tour has a duration of maximum $10$ hours. These constraints are described in below equations:

- $\forall t \in \{1..N\}, E_t = \sum_{\substack{(i,j)\in\{1..M\}\times\{1..|T_i|\} \\ T_{i,j}=t}} S_{i,j}$
- $\forall t \in \{1..M\}, \sum_{i=2}^{|T_t|} D_{T_{(t,i-1)},T_{(t,i)}} + D_{0,T_{(t,1)}} + D_{T_{|T_t|},0} + 30 \times \sum_{i=1}^{|T_t|} S_{t,i} \leqslant 600$

### 1.2.4 Optimization criteria

One solution $(T1, S1)$ is better than another $(T2, S2)$ if $|T1| < |T2|$. If $|T1| = |T2|$, then $(T1, S1)$ is better if and only if the sum of total duration of all tours in $T1$ is smaller than the one of $T2$. The sum of $S$ - examination time is irrelevant in comparing because it is constant. Other optimizations criteria can be considered, for example reducing work hours imbalance by days or using a smaller number of distinct *routes* but for now these are left for future work.

### 1.2.5 Particularities relative to classic problems

As mentioned in the section 2 there are other well-known problems in the literature which seem similar to the one that we elaborated a solution. Under the definition above, our problem has the following particularities:

- *minimum number of vans/vehicles/salesman* or agents in general: is most important optimization criteria. This is far less frequently meet in other papers.
- *multiple visits of each township*: nodes have a number of homogeneous *tasks* that have to be done by agents, which consume time. Passing through a node multiple times is allowed even without doing any task.
- *time constraints on tour length*: all routes represent working days that are limited to a number of hours. This is easier to handle than the more frequent *time windows constraints*, and in our case reasonable for patients, as eventually, time preferences of patients can be satisfied by a different scheduler algorithm.
- *all tours start and end in the capital*, known as *single depot* in vehicle routing.

### 1.2.6 Conclusion

- The examinations periodicity that was specified in initial user requirements even if it seems like a challenge was solved by how the problem was modeled;
- During the project implementation the number of examinations may vary. The number that is currently used is just an estimate of the number of examinations per month and it can be easily changed since it is part of the input data;
- The minimum number of required vans will be obtained when you take the minimum number of tours divided by 21 and round it to the higher integer.

## 1.3 Related Problems

The problems presented in the following section have similar features to our problem.

### 1.3.1 Traveling Salesman Problem (TSP)

The *travelling salesman problem* (**TSP**) asks the following question: *"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"* It is an **NP-hard** problem in combinatorial optimization, important in operations research and theoretical computer science. [1]

### 1.3.2 Multiple Traveling Salesman Problem (mTSP)

The *Multiple Traveling Salesman Problem* (**mTSP**) is a generalization of the Traveling Salesman Problem (**TSP**) in which *more than one salesman is allowed.* Given a set of cities, one depot where $m$ salesmen are located, and a cost metric, the objective of **mTSP** is to determine a tour for each salesman such that the total tour cost is minimized and that each city is visited exactly once by only one salesman. [2]

### 1.3.3 Vehicle Routing Problem (VRP)

The *vehicle routing problem* (**VRP**) is a combinatorial optimization and integer programming problem which asks *"What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?"*. It generalizes the well-known travelling salesman problem (**TSP**). It first appeared in a paper by George Dantzig and John Ramser in $1959$, in which first algorithmic approach was written and was applied to petrol deliveries. Often, the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. The objective of the **VRP** is to minimize the total route cost. In $1964$, Clarke and Wright improved on Dantzig and Ramser's approach using an effective greedy approach called the savings algorithm. [3]

The objective function of a **VRP** can be very different depending on the particular application of the result but a few of the more common objectives are:

- Minimize the global transportation cost based on the global distance travelled as well as the fixed costs associated with the used vehicles and drivers

- Minimize the number of vehicles needed to serve all customers

- Least variation in travel time and vehicle load
- Minimize penalties for low quality service

### 1.3.4 Periodic Vehicle Routing Problem (PVRP)

The **PVRP** is a generalization of the classic *vehicle routing problem* in which vehicle routes must be constructed over multiple days (we use *"day"* as a general unit of time throughout this chapter). During each day within the planning period, a fleet of vehicles with limited capacity travels along routes that begin and end at a single depot. The underlying graph $G = (N, A)$ is assumed to be a complete network with known travel costs along the set of arcs, $A$. The set of nodes, $N$, includes the depot and customers that are visited with predetermined frequency over the planning period. The objective of the **PVRP** is to find a set of tours for each vehicle that minimizes total travel cost while satisfying operational constraints (vehicle capacity and visit requirements). [4]

## 1.4 Related Work

This section aims firstly to present similar problems that computer scientists try to solve and secondly it aims to reveal the impact that similar initiatives had on *infant mortality rate*.

### 1.4.1 Related Problems in Computer Science

The problem that we try to solve can be found in many other domains. It can be envisioned as a particular case of any sort of periodical maintenance or general tasks at given locations around a central node, that is the residency of working agents.

There are well-known problems that can be related to ours such as *Traveling Salesman Problem* (TSP) and *Vehicle Routing Problem* (VRP). Being so well know and studied brought to many variations, a part of them containing similar constraints and features to the ones that we encounter.

For example *Multiple Traveling Salesperson Problem* (mTSP) contains one of the most important particularities for the problem that we try to solve, both of them have to use multiple vans in order to achieve their goals. Almost all the mTSP versions work with the same number of vans and usually the goal is to optimize either the distance that they are traveling or the cost, but there is one version of the mTSP named *Crew scheduling* which requires minimizing the number of vans, this being the goal that we set at the beginning of this project and also this is one of the most important particularities of our problem. Another key difference that our problem has is that in order to complete the task a node can be transited multiple times on the same route, with or without scheduling investigations during visits.

Another version of a well-known problem, VRP, that shares similarities with ours is *Period Vehicle Routing Problem* (PVRP). First one is that it involves periodic deliveries, which in our case are medical services. The second one is that it requires the optimization of the fleet size, this would be translated in our problem as minimizing the number of vans. Although it shows similarities to our problem, PVRP contains differences among which there are remarked the following:

- that no time is spent on one delivery;
- periodicity can differ on each customer;
- vehicles have capacities.

It has been proven that TSP, VRP and their variations are in the complexity class NP-hard. This implies that there is no polynomial deterministic algorithm that that produces global optimum output for the problems mentioned above. Researchers showed a huge interested in studying these problems because of the multiple real-life scenarios that they can be applied to. Due to the high interest, a lot of algorithms generate a good approximation of the result has been developed. Some of the solutions have been proven to provide outputs that are very close to the global optimum. The methods to approximate the solution that researches developed are touching different fields of computer science among which we find genetic algorithms, tabu search, simulated annealing, other bio-inspired optimization. Some approaches involve machine learning or distributed algorithms.

### 1.4.2 Related Work in Healthcare

Ultrasound can diagnose many of the most common causes of maternal and neonatal mortality. Many studies have emphasized the importance of prenatal testing especially providing obstetric ultrasonography which has been proven that improves the infant (or neonatal, perinatal or maternal) mortality. Besides, that other benefit of prenatal medical investigations is an early observation of less severe abnormalities, that are not life-threatening and sometimes can be easily treatable.

Uganda is one of the countries with the highest IMR. An experimental project has been conducted starting from 2010 in the rural areas in Uganda that aims to lower the IMR and it turned out very successful, which was an expected result because of high IMR.

Medics Caravan is a Romanian project that started in 2014 and aims to help the people that are in need and can't access proper medical care, people that live in isolated rural areas in Romania. In order to be aware of the issues that can occur while implementing our project meetings with medics that founded Medics Caravan were set up. Even if it does not seem to have any importance from the computer science perspective, which handles only ensuring the visits schedule, it helped us to set our expectations to the practical challenges that we can encounter. Two of the most important ones that were revealed during the meeting was announced a visit to a remote village and if it is feasible to rely on local electric power sources.

# Chapter 2

# Used Concepts and Technologies

## 2.1 Simulated Annealing

*Simulated annealing* is a meta-heuristic successfully used in many applications for finding (or rather, approximating) the global optimum of a function. It starts with an initial random solution, and, at each time step, it generates a neighbor solution, evaluates it, and decides to move to it or not based on which one is better, and by how much. The algorithm first has a high probability of accepting worse solutions, that gradually decreases over time; this is designed to simulate the process of annealing in metallurgy, hence the name of the algorithm.

## 2.2 Genetic algorithms

A genetic algorithm is a meta-heuristic inspired by the natural process of evolution of species. A population of individuals, also known as chromosomes, each one representing a solution, is evolved during multiple generations.

At each generation, the new population is **selected** from the old one, after some individuals undergo **crossover** (exchanging parts of the solution representations between 2 individuals), and **mutations**, that randomly change the underlying solution. Obviously, the chromosomes with a higher score have a higher probability of being **selected** for the next generation, although lower score solutions are necessary for preserving the variance of the population, especially during the later generations.

## 2.3 DEAP Library

**DEAP**, *Distributed Evolutionary Algorithms in Python*, is a Distributed Evolutionary Algorithm (EA) framework written in Python and designed to help researchers developing custom evolutionary algorithms. Its design philosophy promotes explicit algorithms and transparent data structures, in contrast with most other evolutionary computation softwares that tend to encapsulate standardized algorithms using the black-box approach. This philosophy sets it apart as a rapid prototyping framework for testing of new ideas in EA research. [5]

# Chapter 3

# An Heuristic Algorithm

The algorithm described in this section is what I have found to give the best results on the given dataset. The main structure of the algorithm came when I tried to devise a simple mechanism to make sure the vans will follow relatively *good* paths, and that proved to be more than enough for a very good solution.

## 3.1 Architecture

The classes used in the algorithm are presented as a diagram in Figure 3.1.

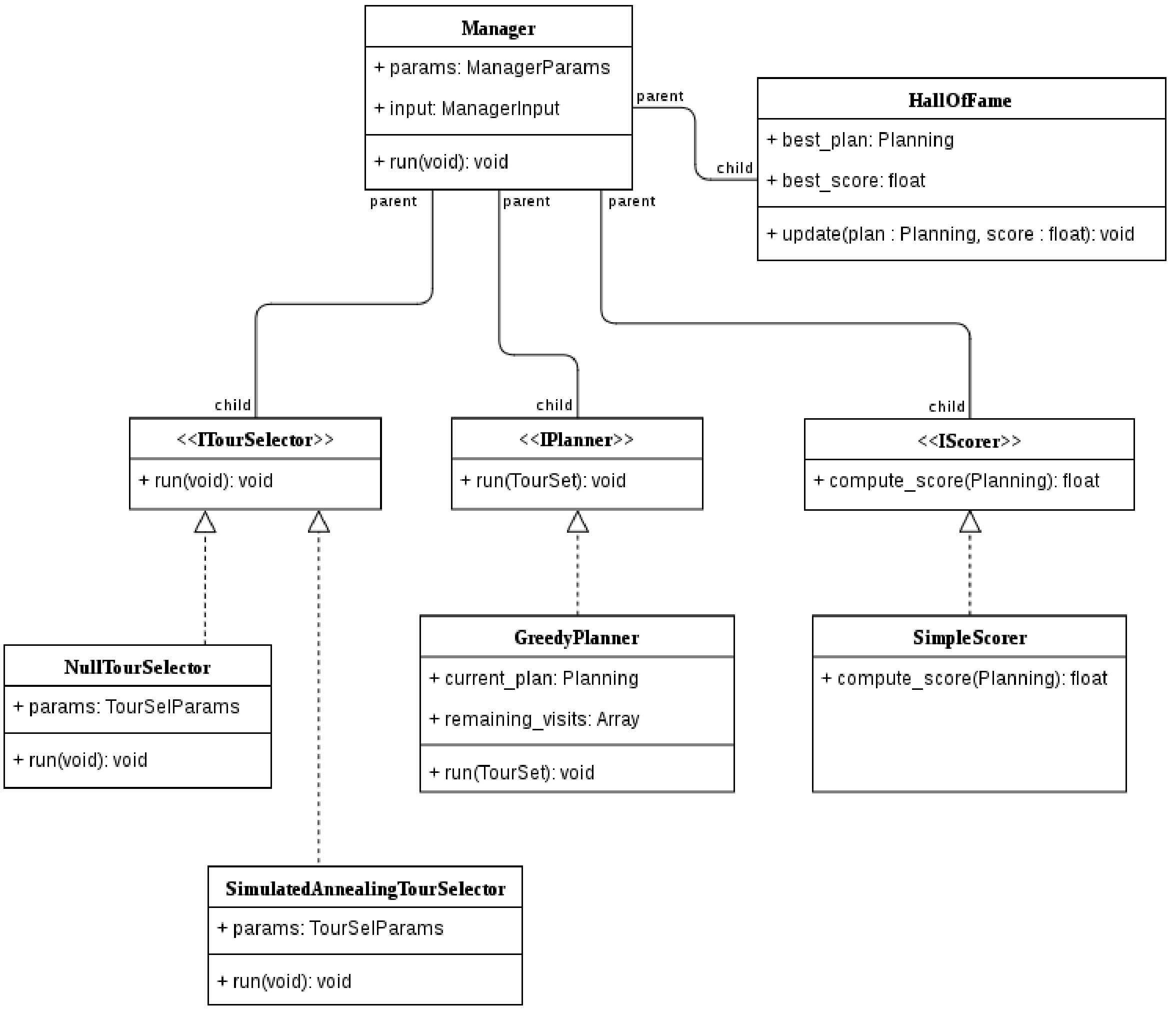


Figure 3.1: Class diagram

Description of the presented classes:

- **Manager** - The main class responsible for instantiating the other classes, based on the input parameters, and running the entire algorithm.
- **SimulatedAnnealingTourSelector** - The first implementation of the tour selector, initially designed to only select a subset of the available tours with a simulated annealing algorithm, and only passing those forward to the Planner. The score of a subset is computed based on the average score from multiple runs of the Planner.
- **NullTourSelector** - After the observation that the restriction of the tours to a smaller subset doesn't improve the quality of the generated solutions, this is the selector currently used for the best results. It sends all the relevant tours to the Planner, every time, and is responsible for the stopping of the algorithm.

- **GreedyPlanner** - Generates a complete planning, using the greedy strategy presented in Section 3.3. The result is then passed to the Manager and forwarded to the Scorer.
- **SimplePlanner** - Computes the score of a planning, as described in Section 3.4. The Manager then uses the result and forwards it to the Hall of Fame.
- **HallOfFame** - Keeps the top generated solution, and also displays the necessary logs.

A run of the entire mechanism is described using the sequence diagram in Figure 3.2.

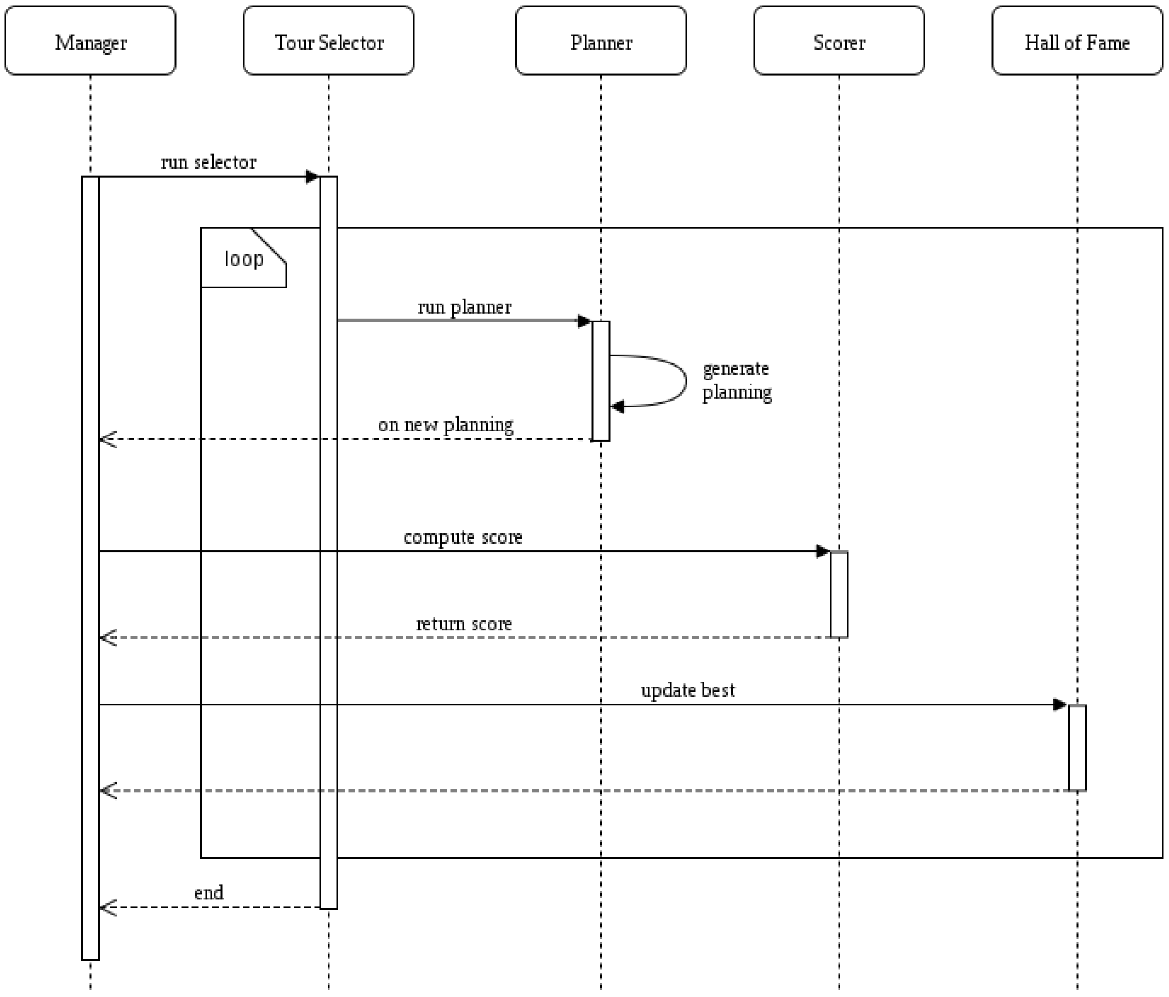


Figure 3.2: Sequence diagram

The steps presented in the diagram are as follows:

1. The Manager runs the Tour Selector.
2. In a loop, the selector calls the Planner in order to generate a complete scheduling.
3. Through a callback, the new planning is sent to the Manager.

4. The Manager runs the Scorer on the newly created planning.
5. The Manager calls the update method on the Hall of Fame with the planning and it's associated score.
6. The Hall of Fame updates it's solutions if necessary, and displays the requested output.
7. When the Tour Selector ends it's loop, the flow returns to the Manager and the algorithm is ended.

The main steps of the entire algorithm, are as follows:

1. Generate a relatively large number of small tours that the vans could follow.
2. Create a complete scheduling by assigning vans to the tours, and performing a certain number of examinations in every visited township.
3. Score a given scheduling based on certain parameters.

## 3.2 Tour generation

This step is done *before* the actual run of the algorithm presented above, and it's results are stored to be used as input for the rest of the mechanism. Obviously, the algorithm can be ran only once, and it's results are reused.

The base tours that have to be generated themselves have to be *optimal* with respect to the number and order of visited communes. Also, it is preferred to be easy to select a subset that can cover all communes, but every tour should also be short enough to be covered in one working day, including any potential examinations. Generating such a set can be seen as a relaxation of the **mTSP** problem, as we allow overlapping tours. However, given that the distribution of examinations is not known at this point in the algorithm, multiple solutions to a **mTSP** problem will be generated, and all the relevant small tours that have been generated will be considered. By relevant we refer to tours that have a chance to be used in the final solution, meaning they are short enough. The quality of the small tours is obviously given by the used **mTSP** solver.

As previously stated, any solver for the **mTSP** problem can be used. It is not always necessary that better **mTSP** solutions will lead to an improvement in the overall result, as more variance in the small tours is more important than generating the global optimum to the **mTSP** instance.

My solution uses a simple simulated annealing procedure to generate the **mTSP** solutions.

## 3.3 Scheduling generation

As input for this stage of the algorithm, we are given the set of available base tours that we can use, and the required number of examinations to be performed in every commune.

The algorithm works as follows. As long as there are still communes that need to be visited, we choose a tour, together with the associated examinations. Then we remove those from the required remaining examinations of corresponding townships.

The choice of the tour is made through a statistical selection. A score is computed for the choice of every possible tour, based on the number of examinations that can be done in that tour (the computation of which can be seen in Algorithm 1), their importance, and the total traveled distance. The entire scoring mechanism is presented in Algorithm 2. Then, a tour is selected with a weighted random from the top $X\%$ (every tour has a probability of being selected directly proportional to its score, and the $X$ is a parameter). This mechanism is described in Algorithm 3.

The choice of which communes should have scheduled examinations during a given tour could be achieved through multiple strategies:

- closest first
- furthest first
- most relevant first
- randomly

The strategies also incorporate a mechanism to limit the number of *single examinations* (traveling to a commune, especially a faraway one and only being able to do one or two examinations). This situation can happen because there are that few remaining examinations to be done in that specific commune, or the remaining time during the working day prevents a longer stay. Ideally, this should be avoided, since it generates longer travel distances.

The variant used in the best results algorithm is the "furthest first" strategy (as shown in Algorithm 1, and the entire mechanism for generating a complete planning is described in Algorithm 4.

**Algorithm 1:** Compute Examinations

```
Input: t, the tour, and E[], the required number of examinations per township
Output: An array of the same length as the input tour, where for every element, the value
        represents the number of examinations to be done in that certain township.

Sort all townships in current tour, furthest from origin come first;
Set the array of examinations to be done as empty;
while the time required to go through the currently selected visits is still smaller than the
  maximum working time per day do
    Select the next township and, if the remaining time allows it, add one examination to
      it;
    Add as many examinations as possible to this specific township, until all necessary
      ones are completed, or there is not enough time remaining;
end
Return the computed array of examinations;
```

**Algorithm 2:** Compute Tour Score

```
Input: t, the tour, and E[], the required number of examinations per township
Output: A single float value, the score of the tour.

Compute the examinations to be done by following tour t;
Return the score as the total number of examinations, divided by the total distance (from
  a greedy selection perspective, this maximizes examinations done per distance unit);
```

**Algorithm 3:** Choose Tour

```
Input: T[], the list of available tours, and E[], the required number of examinations per
       township
Output: A single tour from T, to be selected at the current step

Compute scores for each tour;
Sort tours descending by score;
Keep only top 10% tours;
Choose randomly from remaining tours, with probability proportional to score;
```

**Algorithm 4:** Generate Planning

```
Input: T[], the list of available tours, and E[], the required number of examinations per
       township
Output: P[], a planning, each element consisting of a tour and examinations per tour

P = [];
while $\exists E[i] \neq 0$ do
    Choose t as the tour to follow at the current step;
    Compute where and how many examinations should be done, throughout the tour t;
    Remove all new examinations from the remaining ones;
    Add tour t and the associated examinations to the output planing P;
end
return P
```

## 3.4 The score of a scheduling

The score is computed as a weighted sum of the following factors, the relevance of each one is a parameter:

- the number of tours that have to be done is the most significant factor;
- total distance traveled, which translates to fuel consumption;
- the number of distinct tours used, by minimizing this metric, the drivers can learn them by heart, simplifying the real-world application of the entire process

# Chapter 4

# Genetic Algorithms

## 4.1 Introduction

I implemented two genetic algorithms, in order to have other solutions to compare my heuristic with, i.e. see if the problem instance is solvable by a smaller number of routes. For this I chose the genetic algorithms, as they are popular and commonly used in solving similar problems. In the following section, I will refer to these two different implementations as *Fixed Tours Genetic Algorithm* (**FTGA**) , and *Greedy Tours Genetic Algorithm* (**GTGA**) respectively.

## 4.2 Complete representation of a solution

A complete solution would have to encode a list of tours that the vehicles have to follow, and the number of examinations that have to be performed at each commune along the way. This representation is quite complex for a genetic algorithm to manage, as the required operators, especially the crossover operator, would have to be very complicated in order to preserve the validity of the solutions. Because of this, the input has been adapted to a more suitable representation.

## 4.3 The usage of the permutations

In the classical **mTSP** problem, each location must be visited by a salesman exactly once. In order to use the well-known permutation approach, the input will be transformed, by duplicating the locations that we need to visit multiple times, instead of having to count the number of examinations in each one. As such, we can represent an order of solving every visit by such a permutation.

Let $P$ be the permutation defined above, and $h$ be the county capital (the start and end point for every tour). The simplest way would be to assign, for each tour $T_i$, a sequence $P_l, P_{l+1}, ..., P_r$ in the permutation, such that the examinations during the tour will be performed in the following order: $h, P_l, P_{l+1}, ..., P_r, h$. Note that there will be no examinations performed in $h$, and that the locations $P_i$, although distinct examinations, can refer to the same commune, as previously defined.

This representation would be complete if the goal was to have only one vehicle go through all the required visits. However, we need a strategy to split the permutation into multiple tours. This strategy is the main difference between the two algorithms.

## 4.4 FTGA tours split strategy

In the *fixed* variant, the limits of the tours are encoded in the representation of the individual, as a list of endpoints, and evolved together with the permutation. This approach has the advantage of having the entire solution represented in an individual, and is the more natural idea. It is also commonly used in applications that solve similar problems.

## 4.5 GTGA tours split strategy

In the *greedy* variant, the permutation is the only information represented in the individual. The limits of the tours are computed in a greedy fashion every time they are required. Basically, we keep adding items from the permutation to a tour as long as there is enough remaining time; when there is not enough, we begin a new tour.

This method is very simple, and defined as follows:

**Algorithm 5:** Generate Tour Assignment

```
Input: P[], the permutation of all examinations
Output: T[], a list of tours that result from permutation P
Let T be empty;
Set the current tour as empty;
foreach examination in P do
    if there is enough time remaining for examination to be added to current tour then
        Add examination to current tour;
    else
        Add current tour to T;
        Reset current tour;
        Add examination to current tour;
    end
end
if current tour is not empty then
    Add current tour to T;
end
return T
```

What follows is the proof that this greedy method of splitting the tours from the permutation cannot generate more tours than the optimal solution (given a set permutation). Obviously, it is possible that an assignment with a shorter total distance exists, but remember that the main objective is to minimize the total number of tours.

Let us assume that the permutation $P$ is divided by the greedy algorithm in $k$ tours, delimited by indexes (inclusive endpoints) $g_0, g_1, g_2, ..., g_k$. For simplicity, $g_0$ and $g_k$ are the first, respectively last indexes of the permutation. Consider that the tours are: $\{(P_{g_0+1}, P_{g_0+2}, ..., P_{g_1})$, $(P_{g_1+1}, P_{g_1+2}, ..., P_{g_2})$, ..., $(P_{g_{k-1}+1}, P_{g_{k-1}+2}, ..., P_{g_k})\}$.

Let the optimal delimitation split the permutation into $h$ tours, where $h < k$. Denote this "best" delimitation as $b$, such that the tours are as follows: $\{(P_{b_0+1}, P_{b_0+2}, ..., P_{b_1})$, $(P_{b_1+1}, P_{b_1+2}, ..., P_{b_2})$, ..., $(P_{b_{h-1}+1}, P_{b_{h-1}+2}, ..., P_{b_h})\}$. Similarly, $b_0$ and $b_h$ are the first, respectively last indexes of the permutation.

Let $w$ be the smallest value for which $b_w > g_w$, $w \in \{1..h-1\}$. This implies $\forall i \in \{0..w-1\}, b_i \leq g_i$. For sure, such a $w$ exists because $h < k$. Given that $b_{w-1} \leq g_{w-1}$ and $b_w > q_w$ the following two cases can be distinguished:

1) $b_{w-1} = g_{w-1}$: the greedy choice ensures that $g_w$ is the highest value for an index in permutation that will give a valid tour that starts with examination $P_{g_{w-1}}$. This implies that adding another examination will generate an invalid tour, and so, the ”optimal” solution would be invalid.

2) $b_{w-1} < g_{w-1}$: in this case the $w$-th tour of the optimal solution will contain following examinations: $P_{b_{w-1}}$, $P_{b_{w-1}+1}$, ..., $P_{g_{w-1}}$, $P_{g_{w-1}+1}$, ..., $P_{g_{w+1}-1}$, $P_{g_{w+1}}$ ..., $P_{b_w}$. It is obvious that this tour is the same as the $w$'th tour in the greedy, with some locations added before and after. The ones added as prefix might generate a valid tour, but the ones added after make this case reduce to the first one: it's impossible to extend the greedy tour any further.

Cases 1) and 2) conclude that the assumption is, indeed, *false*, which implies that the greedy algorithm generates a solution that has at most as many tours as the optimal solution (given the same input permutation).

## 4.6 Initialization of a chromosome

### 4.6.1 Random initialization

The first attempted initialization was completely random. Basically the permutation was initialized with a random shuffle of the numbers from one to the number of required visits. This proved quite ineffective for the genetic algorithm, as all the indexes representing the same location are spread out, and it is hard for the algorithm to group them together that well.

### 4.6.2 Smart initialization

An alternative, ”smart” initialization, keeps the indexes associated with the same location together, and only randomizes the order of the such formed ”groups”. This apparently small change drastically improves the performance of the genetic algorithms, making them almost as good as the heuristic solution. This can be seen best in Chapter 5 (results).

## 4.7 Mutation of a chromosome

Three types of mutation are being used by both **FTGA** and **GTGA**, for the permutation part:

- swap two random positions from the permutation

- reverse a random subsequence from the permutation
- shuffle a random subsequence from the permutation

For **FTGA**, random increments, decrements, or randomizations in a single endpoint at a time are also used.

The mutation that should be used each time is chosen randomly based on a distribution given as parameter.

## 4.8 Crossover of two chromosomes

There are three crossovers being used by both genetic algorithms for the permutation part:

- Ordered Crossover (OX) [6]
- Partially Matched Crossover (PMX) [7]
- Uniform Partially Matched Crossover (UPMX) [8]

Similarly to how the mutations are chosen, each time a crossover is required, it is sampled from a random distribution.

## 4.9 Evaluation of a chromosome

The evaluation is obviously different for the two types of algorithms, so both will be described next. The greedy one is mentioned first, since it is the simpler one.

### 4.9.1 GTGA evaluation

The evaluation for this case is straightforward; once the tours are selected from the permutation in the aforementioned greedy manner, the score of a chromosome is given by the sum following factors:

1. the number of tours, multiplied by a factor;
2. the total distance traveled.

Obviously, we want the factor to be large enough such that a solution with fewer tours but higher total distance will always be preferred to one with more tours. This forces the optimization of the number of vehicles to come first, as they are the most expensive resource.

### 4.9.2 FTGA evaluation

As the tours are already separated, but they can be invalid, a different metric had to be used in this scenario. Of course the total distance is considered, but the main goal to get the total ”extra time” (time beyond the working day that a tour would require) to zero. This can be interpreted as an error that has to be minimized, and we do this by taking the *sum of squares* of these errors and using it instead of the number of tours mentioned in the **GTGA** evaluation.

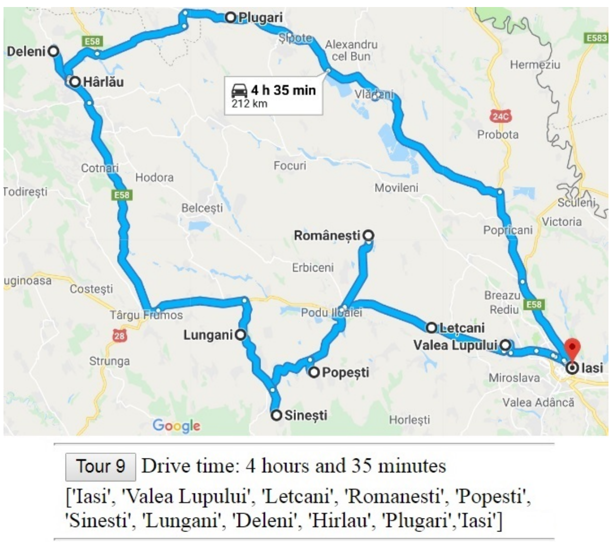
# Chapter 5

# Results and Comparisons

## 5.1 Visualization of routes and scheduling

A method to visualize the resulting tours was also implemented. The scheduling is provided in simple text format, for each of the 21 days, tours are printed for all vans, with the basic tour numbers and total driving time, examination time, their sum (giving total work time for that certain day), and townships in visiting order with the number of scheduled examinations in each one. From this, it is easy to produce a schedule for each township separately to see on what days and hours patients have appointments. Patients schedule preferences is out of our focus now and would probably be a very difficult task to collect this information.

Tours are of interest especially for drivers. All basic tours (tours generated in the first part of the heuristic algorithm) are included in a web page, where each tour has an associated button loading a Google Maps frame displaying that tour, as shown in Fig. 5.1.



Tour 9 Drive time: 4 hours and 35 minutes
['Iasi', 'Valea Lupului', 'Letcani', 'Romanesti', 'Popesti', 'Sinesti', 'Lungani', 'Deleni', 'Hirlau', 'Plugari','Iasi']

Figure 5.1: Example visualization of a basic tour

## 5.2 Comparisons between multiple implementations

Table 5.1 displays the results of the three algorithms presented above on our dataset.

Table 5.1: Average runs results

| **Algorithm variation used** | **Time limit in seconds** | **Average number of tours** | **Average total distance** |
|---|---|---|---|
| 1) default heuristic algorithm | 20 | **42.2** | 369905 |
| 2) closest locations first | 20 | **42.8** | 393990 |
| 3) ratio tour score computation | 20 | **44.0** | 385327 |
| 4) larger keep percent | 20 | **42.2** | 379067 |
| 4) larger keep percent | 40 | **42.0** | 377200 |
| 5) GTGA[1]random initialization | 60 | **82.6** | 764420 |
| 5) GTGA[1] random initialization | 3600 | **63** | 565800 |
| 6) FTGA[1] random initialization | 600 | **75** | 635025 |
| 7) GTGA[1] smart initialization | 20 | **44** | 420113 |
| 7) GTGA[1] smart initialization | 300 | **43.7** | 398010 |

## 5.3 Descriptions of presented algorithm variations

1) The **default heuristic algorithm** uses the following parameters:

   - selects the furthest locations first when choosing the townships to visit in every tour, as presented in Algorithm 1;
   - the score of tours is computed as described in Algorithm 2, that is the difference between the number of examinations made and the total distance traveled (the number of examinations is multiplied by a factor, in order to preserve relevance);
   - the *keep percent* is set to 20% (how many top tours should be considered relevant when scoring them); this parameter is used in Algorithm 3.

2) The **closest locations first** algorithm drastically changes the order in which examinations are selected along a tour. This seems to give worse results, due to the fact that there are multiple faraway examinations to be made in the last days, and they are not grouped well, as they would have been in they were done earlier during the scheduling.

[1]GTGA and FTGA are the genetic algorithms defined in Section 4.1

3) The **ratio tour score computation** algorithm uses ratio instead of subtraction; basically computes the score as the ratio between number of examinations done and total distance, instead of a subtraction (in the subtraction case, the number of examinations is multiplied by a factor, in order to preserve relevance). Even though this alternate method sounds more logical in theory, it doesn't seem to give any better results.

4) The **larger keep percent** algorithm increases the *keep percent* to 40% (up from 20%), thus improving the search space of the algorithm, although increasing the running time for finding a good solution. By making this value even larger the algorithm would theoretically find even better solutions, although, in practice, because of the randomness and the time restrictions, it is not feasible.

5) The **GTGA** uses a $\mu + \lambda$ strategy [9], with $\mu = 150$ (population size), $\lambda = 300$, a crossover probability of $0.6$ and a mutation probability of $0.2$. These parameters were chosen empirically after multiple runs.

6) The **FTGA** uses the same parameters as **GTGA**, and the genetic operators used in both algorithms are described in Section 4

7) The **smart initialization** is what greatly improves the results of the genetic algorithm, as seen in the results. This method is described in Section 4.6.2.

# Chapter 6

# Future work

## 6.1 Directions

Currently, there are 3 future work directions:

1. Improving algorithm performance, regarding both speed and generated solutions.
2. Improving the general usability of the algorithm, adapting it to other scenarios that might come up during the real-life implementation of the project.
3. Actually starting the project, acquiring the vehicles, equipment, employing medic specialists and drivers; this direction is currently blocked because of the lack of funding.

## 6.2 Algorithm performance

Regarding the algorithm, a simple, but very promising way to improve the quality of the solutions would be to generate multiple good, but diverse solutions using the heuristic algorithm, and then initializing the first population of the genetic algorithm with such individuals. Given enough variance in the generated set, the genetic algorithm can greatly improve already very good solutions. The way variance can be preserved is by not using permutations that are very ”similar” to each other. Indeed, this is a very vague metric, but it can be tested. Here are some methods that could check such a similarity:

- Counting the number of identical positions
- Computing the maximum / average length of maximal common subsequences

- Counting the number of required inversions to change one permutation to another (a metric similar to the Levenshtein distance)
- Any weighted combination of the above methods

## 6.3 Project implementation

As for project implementation, the next phase is obtaining funding for the initial acquisition of at least two equipped vans, and finding drivers and medic specialists. The algorithm implementation is not blocking the project currently. For now, it may be more important to change how the problem is modeled, what practical aspects are translated into input data and what are the most appropriate optimization criteria. It is not hard to adapt our application to many such changes, but it is harder to imagine all the possible ones before the project real-life implementation begins. After real drivers and medics start to cover scheduled routes we expect that they will provide us with all the necessary feedback and requests for new features, and prove that tours are doable in the time intervals we assign them, or the contrary.

Simple changes can be done by modifying parameters, score functions and even the returned solution structure, to some extent. More elaborate changes would be required if we would tackle the following, possibly useful aspects: managing data of pregnancy stages per patient level, managing appointments per patient with the possibility of rescheduling on missing appointments, including smaller settlements than communes, or real-time re-scheduling of examinations when unexpected events are encountered in the field (for example, traffic jams).

# Conclusion

I believe that I have reached the initial goal of finding an approximation for the number of vans that are needed initially for developing the project and visualizing the generated tours and scheduled appointments for each day of the month, through the Google Maps API.

From the observed results, two vans seem to be required, according to the provided data, as a complete planning with 41 tours per 21 days has been generated. As a very relaxed lower bound, if we consider the number of required examinations, 583, and that each one takes 30 minutes, the total time spent on the investigations alone is equivalent to 30 working days, for a 21 days period (a working day in our case was considered to have 10 hours). This proves two vehicles is the minimum solution for this problem instance.

The application of these algorithms has shown that a more complex problem than a human can solve is easy to tackle by implementing such solutions. Moreover, it is easy to reschedule the entire planning if any configurations change, by applying the same algorithm to different regions as well, or adapt it to include more input elements or restrictions.

# Bibliography

[1] Wikipedia. Travelling salesman problem — Wikipedia, the free encyclopedia, 2018. [Online; accessed 26-June-2018].

[2] NEOS Guide. Multiple traveling salesman problem (mtsp), 2018. [Online; accessed 26-June-2018].

[3] Wikipedia. Vehicle routing problem — Wikipedia, the free encyclopedia, 2018. [Online; accessed 26-June-2018].

[4] Peter M. Francis, Karen R. Smilowitz, and Michal Tzur. *The Period Vehicle Routing Problem and its Extensions*, pages 73–102. Springer US, Boston, MA, 2008.

[5] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[6] Lawrence Davis. Handbook of genetic algorithms. 1991.

[7] David E Goldberg, Robert Lingle, et al. Alleles, loci, and the traveling salesman problem. In *Proceedings of an international conference on genetic algorithms and their applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[8] Vincent A Cicirello and Stephen F Smith. Modeling ga performance for control parameter optimization. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 235–242. Morgan Kaufmann Publishers Inc., 2000.

[9] Aram Ter-Sarkisov and Stephen R Marsland. Convergence properties of ($\mu$+ $\lambda$) evolutionary algorithms. 2011.