UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

PleasantTourFinder : An application to find the most pleasant tour from a given location

Dissertation presented by
Damien MERCIER

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor
Pierre SCHAUS

Readers
Michael SAINT-GUILLAIN, Yves DEVILLE

Academic year 2015-2016

Contents

Abstract

When you are in an unknown region, it happens that you would like to take a walk that comes back to your starting location (e.g. where you left your car). Because you don't know the region, it is difficult to find a beautiful and pleasant tour of the good distance. Moreover, web and mobile applications like Google Maps are mainly based on the shortest or the fastest path and thus fail at providing easily routes that are pleasant. The goal of this master thesis is to create an user-friendly application allowing everybody to search for a pleasant tour corresponding to his own preferences. We divide the problematic of finding the most pleasant tour in two distinct sub-problems. First, we define the routing optimization problem and we solve it using Constraint Programming and Large Neighborhood Search techniques. Second, we tackle the assessment of a *pleasant value* for each road segment by using cartographic data and the preferences of the user. Cartographic data we use are based on the data from OpenStreetMap (OSM) on which we compute characteristics about roads and their environments. Finally, we implement all needed components resulting in a fully functional web application that allows user to search for a pleasant tour. Experimentations validate some expectations but also point out some critical parts that can be improved in a further work.

Acknowledgments

At the end of this master thesis, I would like to express my sincere gratitude to my supervisor Professor Pierre Schaus and to the assistant Michael Saint-Guillain for their supervision during the conception of this project. I would especially like to thank Michael Saint-Guillain for the useful comments, his availability and his engagement throughout the year, and to Professor Schaus for the opportunity he gave me in preparing this project and his confidence.

I would also like to thank my loved ones for their support during all the writing process of this master thesis and particularly Luc Mercier and Louise Willocx for their attentive reading.

Introduction

In our everyday life, it sometimes happens that we want to do a ride but we do not have always a sufficient knowledge of the region in order to plan it. Imagine for instance that you have rent a beautiful chalet for a weekend with some friends and you would like to go for a walk. In this case, you can take a map of the region and choose manually which roads and tracks you want to follow. Nevertheless, the required choices are hard to devise and it is difficult to obtain a tour of the desired distance. Current online routing services like Google Maps and its concurrents do not propose to make routes that have the same starting and ending point. Indeed, classic routing services are based on the shortest path or on the fastest path. When we want to do a walk, the intent is to follow pleasant paths and to come back to our initial location. Ideally, we do not want a basic "round-trip", but rather a cycle shaped trip. Moreover, we would like that the tour fits to the desired time or to our physical condition, restricting the total length of the tour. We have mentioned an example where we want to do a walk with friends but it exists in fact others situations where people need a pleasant tour constrained by the starting location and the total length. For instance, this kind of pleasant tour can be very useful for all people that do outside sports likes runners, hikers, cyclists and horsemen.

Our application gives you all of this. A user starts by visiting our web interface. Then he can select from where he wants to start his tour and the distance that he would like. He defines also his mode of transportation and his preferences about what he appreciates when he walks. Finally, he can launch the search and see the tour that suits best his expectations.

We are thus interested in solving the problem of finding a pleasant tour given the constraints over the distance provided by the users and their preferences. This kind of problem is in fact generally decomposed in two sub-problems. The first sub-problem consists of finding a tour near a given distance that optimizes a particular value over the tour. This particular value is assumed computable for each road and can be anything. The second sub-problem consists of assigning a value to asses the pleasure felt by a person when he walk on a particular road. This value can finally be used in the first sub-problem to solve our initial problem. The advantage of splitting the problematic is two-fold. First, each problem taken separately is easier to understand and to resolve. Second, sub-problems are easier to reuse in other works than the more specific global problem.

As pleasant tours are something that should interest the most of the people, solving uniquely the problem is not enough. We are also interested in providing to people an easy way to find a nice tour. This master thesis is thus dedicated to the creation of an application that allows everyone to find easily a pleasant tour that is near the desired distance. The goal is to create all the components that are needed to provide to the final user a functional and user-friendly application.

In addition with the two sub-problems mentioned before, there are several challenges behind

the application. A first one is to compute a solution of the problem in a reasonable time. Indeed, in today's society, everything must be fast. The user interface is also challenging because the user's expectation and the usages of the application can be multiple. A very simple application has the advantage to be easy to use but risks of frustrating user. While a too complex application has the advantage of proposing all needed parameters to the user but risks also to frustrate users that just want obtain quickly a tour. A last challenge about our application is dealing with the large amount and variety of data that we have at our disposal through Internet.

This master thesis dissertation is organized as follows. In the chapter 1 we tackle the first sub-problem, namely, the routing optimization problem consisting of finding a tour of a given distance optimizing a specific value. For this first problem, we begin in the section 1.1 by introducing the needed knowledge in graph theory. Then, in section 1.2, we introduce routing problems. In section 1.3 we review the literature to see if the problem that we want to tackle has already been handled. We finally define properly our routing problem in mathematical terms in section 1.4. Chapter 2 is dedicated to the second sub-problem, that is the assessment of the pleasure associated to a road segment. We start this chapter by a more precise specification of the problem in the section 2.1. Then we review the literature in the section 2.2 to see which approaches are already used to asses the pleasure or the beauty of a road. After that, we analyse and expose which are the available sources of data and what are their nature. Finally, we define how we actually compute a *pleasant value* for each particular road segment. As the purpose of this master thesis is to create an application, we present in the chapter 3 the final implementation and algorithms used to this aim. We begin in section 3.1 by an overview of the different components that compose and interact to produce our application. Sections 3.2 to 3.6 presents each of these components more in details. Chapter 4 presents some experiments and some results that we have been obtained on our application. Finally, we conclude this master thesis by a conclusion on the work archieved so far and on some elements that can be interesting to study or improve in a further work.

Chapter 1

First problem : The routing problem

As described in the introduction chapter, we have divided our problematic into two distinct problems. The first one is the problem of finding a route that starts from one location and comes back to this point with time or distance as constraint. Moreover we want this route to be the most pleasant possible. It is therefore an optimization problem, based on this "most pleasant" criterion.

Chapter 2 discusses how an algorithm could assess the *pleasant value* of a road segment. This chapter simply considers that this value is available for each road segment. We focus here only on the routing optimization problem.

In this chapter, we clearly define the routing problem. Before we can begin to talk about our problem, we have to present some mathematical bases required to understand the rest of this chapter. We first define some terms, then we introduce routing problems with simple examples. After that, we present problems that are already described in the literature. The end of this chapter is finally focused on our particular problem by defining precisely our routing problem in mathematical terms.

1.1 Introduction to graph theory

When we talk about routing, we must first explain how we represent the map on which we have to find a route. In common words, a map is a schematic representation of a topographic reality. For routing problems, we are mainly interested in the road networks (but a map can contain other useful information as we will discuss in the chapter 2). To easily manipulate this kind of reality, the mathematics define some models including *graph theory* that we introduce here. This introduction is widely inspired from the book *Introduction to graph theory* by West [40] and the reader willing to know more about graph theory can find more information in this book. We limit ourselves here to concepts required to describe our problem and understand the literature associated with it.

In graph theory, the first term to introduce is obviously the concept of *graph*. The common example for that is the example of the *seven bridges of Königsberg*. As shown in figure 1.1a the city of Königsberg has seven bridges over the Pregel river. Each bridge can be considered as a connexion between two land masses. A graph representation of this reality is shown in

figure 1.1b. In the graph, the different land masses (noted from v_A to v_D) are *vertices* and the bridges (noted from e_1 to e_7) are *edges*. Now that we have a first idea of what is a graph, we will see his formal definition.

Definition 1 : Graph

A **graph** G is a triplet (V, E, φ) where

- V is a set of *vertices*,
- E is a set of *edges*,
- and φ is a relation that associates two vertices with each edge.

From this definition we obtain, for the example of the seven bridges, the graph G that contains the set $V = \{v_A, v_B, v_C, v_D\}$, the set $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ and the relation φ which assigns to each edge the vertices corresponding to the endpoints of the bridges (see figure 1.1).

As shown in the figure 1.1b, each vertex is represented by a point and each edge is represented by a line between the two vertices associated to it by the φ relation.

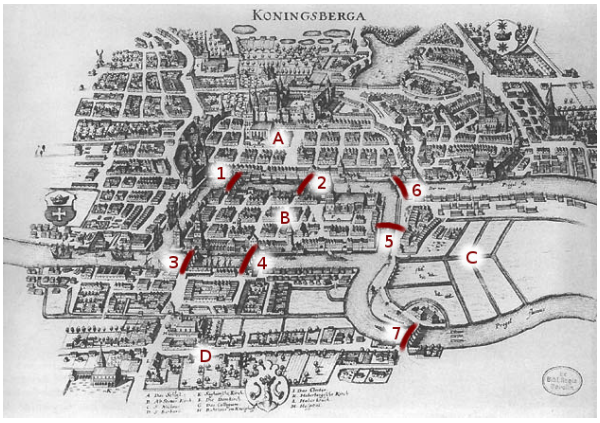
From this concept of graph, we can create the concept of *subgraph*.

Definition 2 : Subgraph

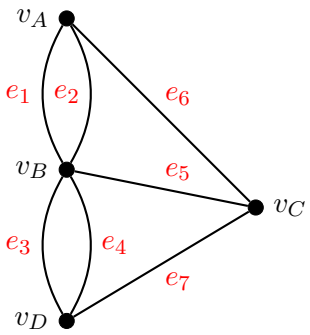
A **subgraph** of a graph $G = (V, E, \varphi)$ is a graph $G' = (V', E', \varphi')$ where

- $V' \subseteq V$,
- $E' \subseteq E$,
- and φ' is the restriction of the relation φ to E'.

For instance, a possible subgraph of graph G, presented in the previous example, is the graph



(a) Historical map (1652)



(b) Graph representation

Figure 1.1: The seven bridges of Königsberg

G' that contains the set $V' = \{v_A, v_B, v_C\}$, the set $E' = \{e_1, e_2, e_6, e_5\}$ and φ' – the relation from φ that assigns two vertices to each edge in E'.

We have also to introduce another type of graph. Let's imagine the case where we have one-way bridges, in this case we can cross the bridge only in one direction. To represent this reality, we have to deal with a *directed graph*.

Definition 3 : Directed graph

A **directed graph** is a graph where the relation φ associates each edge with an *ordered* pair of vertices. The first vertex is the departure point and the second is the arrival point.

In order to easily distinguish the undirected from the directed case we introduce the notion of *arc*.

Definition 4 : Arc

An **arc** is a (directed) edge that has a starting vertex and an arrival vertex.

In this master thesis, we use the term *edge* to refer to the undirected edge and the *arc* when we want to refer to the directed case. Arcs are represented by an arrow from his starting vertex to his arrival vertex whereas edges are represented by a simple line.

Remark. *It is possible to have a graph with both edges and arcs. This is a typical case when we work on a road network with some one-way roads. This mixed case is usually turned to the directed case by putting two arcs with opposite directions in stead of each edge.*

Another important notion for our problem, is the notion of weighted edges (or weighted arcs).

Definition 5 : Weighted graph

A **weighted graph** is a graph where we assign a value to each edge (or arc). This value is called the *weight* of the edge.

In a graph that represents a road network, each edge can for instance be associated to a value corresponding to the length of the corresponding road.

Other properties of graphs are the followings :

Definition 6 : Simple graph

A **simple graph** is a graph without multiple edges nor loops. *Multiple edges* are edges that have the same two vertices as extremities. A *loop* is an edge that has the same vertex at his two extremities.

Definition 7 : Complete graph

A **complete graph** is a simple graph where all pair of vertices are linked together by an edge. (Figure 1.2 is an example of complete graph.)

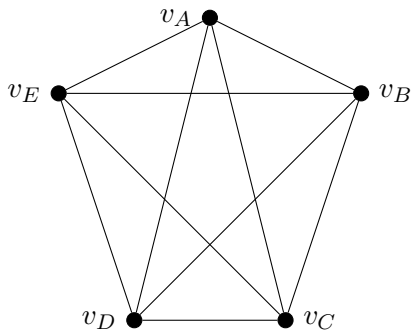


Figure 1.2: Example of complete graph composed of 5 vertices, also known as K_5.

Now that we have an idea of what is a graph and what can be some of his properties, we will talk about how to travel in a graph. Indeed, our work is about the problematic of walking through a road network being represented by a graph.

Definition 8 : Walk

A **walk** is an ordered sequence $(v'_0, e'_1, v'_1, .., e'_n, v'_n)$ of vertices and edges where each edge e'_i has the extremities v'_{i-1} and v'_i. A walk is said **close** if $v'_0 = v'_n$ and **open** otherwise.

For the ease of notations, we denote $(v'_0, v'_1, ..., v'_n)$ a walk in a simple graph.

If we reuse the graph shown at the figure 1.1b, a possible closed walk is the ordered sequence $\{v_A, e_1, v_B, e_5, v_C, e_6, v_A\}$, where $v'_0 = v_A$, $e'_1 = e_1$, $v'_1 = v_B$, and so on. This walk starts from the vertex v_A, then goes to the vertex v_B passing by the bridge e_1, then goes to the vertex v_C passing by the bridge e_5 and finally crosses the bridge e_6 to going back to the vertex v_A.

Let's now introduce some possible properties of walks that can be useful for our problematic: Trail and path.

Definition 9 : Trail

A **trail** is a walk that does not have any repeated edges.

Definition 10 : Path

A **path** is a walk that does not have any repeated vertices. A **closed path** – also named a **tour** – is a path where the two extremities are the same vertex.

In a weighted graph, the **weight** of a walk, a trail or a path is computed by summing the weight of each edge traversed by the walk, the trail or the path.

The definitions of eulerian and hamiltonian paths that are presented below are useful to understand the state of the art of routing problems.

Definition 11 : Eulerian path and graph

A path is said **eulerian** if it visits all edges of the graph exactly once.
A graph is said eulerian if it contains an eulerian path.

Definition 12 : Hamiltonian path and graph

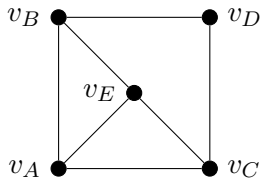
A path is said **hamiltonian** if it visits all vertices of the graph exactly once.
A graph is said hamiltonian if it contains a hamiltonian path.

Recall the example of the seven bridges of Königsberg, a question that we can ask is the following : Are there enough bridges to be able to go from anywhere in the city to any other location? In other words, we want to know if there are not several parts of the city that are isolated from each other (a group of islands without any bridge connecting to the rest of the city for instance). In graph theory, this property is formalized by the concept of *connected graph* :

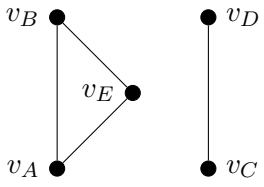
Definition 13 : Connected graph

A graph is **connected** if for all pair of vertices, we can construct a path that link the two vertices. Otherwise, the graph is said **disconnected**

The figure 1.3a shows an example of connected graph while figure 1.3b shows an example of disconnected graph. In this last example we can see that this is not possible to construct a path between the vertex v_A and the vertex v_D.



(a) Connected graph



(b) Disconnected graph

Figure 1.3: Connected and disconnected graph example

1.2 Introduction to routing optimization problems

Now that we have explained the basics of graph theory, we can introduce the problematic of finding a tour into a graph representing a road network. There is a large variety of routing problems. A very popular one is the Traveling Salesman Problem (TSP). This problem is described as follows. A salesman needs to visit several cities in order to sell products. He knows

the cities he wants to visit and he seeks for a tour that minimizes the total distance (or the total time) of his travel, that is, which is optimal in terms of travel costs.

Let's now introduce a formal definition for this problem. Mathematically speaking, we can represent the problem in a graph where each city is a vertex $v \in V$ and each city is linked to other by a route represented by edges denoted by E. We therefore consider a complete graph $G = (V, E)$. Moreover, a weight is assigned to each edge $e \in E$ that corresponds to the distance. This travel cost is noted c_{ij} for the edge connecting the vertex $i \in V$ to the vertex $j \in V$. To formulate the TSP problem, let's now introduce a decision variable x_{ij} for each pair of vertices (i, j) :

$$x_{ij} = \begin{cases} 1 & \text{if the solution contains the edge from } i \text{ to } j \text{ (and } i \neq j) \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

We introduce also the mathematical notation $\delta(S)$ that is the set of all edges that has exactly one of its extremities in S.

The purpose of the TSP is to find an *hamiltonian closed path* that minimizes the total distance. In other words, we search a closed walk that passes exactly once in each city except the starting one where we pass two times (at the start and at the end of the tour) and with the smallest weight. If we consider the problem with n cities, this can be mathematically formulated as follows :

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} c_{ij} \tag{1.2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \tag{1.3}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \tag{1.4}$$

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 2 \qquad \forall S \subsetneq V, S \neq \emptyset \tag{1.5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i \neq j \tag{1.6}$$

The objective of the TSP is defined by the equation (1.2) whereas equations (1.3) and (1.4) force respectively each city to be a departure and an arrival of exactly one edge. Finally, the inequalities (1.5) – called *subtour elimination constraints* – serve to exclude solutions where we have subtours. We take all subsets of the set of vertices V that are not equal to V nor the empty set. Then we count by summing over x_{ij} the number of edges that link a vertex inside S and a vertex outside S. This sum must always be greater or equal to 2 because we want that all subsets are linked together in order to do not have subtour. For instance, if we have six cities denoted v_A to v_F without the constraints (1.5), the solution shown in the figure 1.4a is valid regarding to the constraints (1.3) and (1.4) but this solution contains two subtours – $\{v_A, v_B, v_C\}$ and $\{v_D, v_E, v_F\}$ – that are not connected together. The addition of the constraints (1.5) eliminates this solution. Indeed, if we take the subgraph $S = \{v_A, v_B, v_C\} \subsetneq V$, we have $\sum_{(i,j) \in \delta(S)} x_{ij} = x_{AD} + x_{AE} + x_{AF} + x_{BD} + x_{BE} + x_{BF} + x_{CD} + x_{CE} + x_{CF} = 0 \ngeq 2$. While the solution presented in the figure 1.4b is for its part valid regarding to all constraints including the subtour elimination constraint.

(a) Invalid solution : contains 2 subtours

(b) Valid solution without subtours

Figure 1.4: Example of solutions

Remark. *As explained by Pataki [26], subtour elimination constraint proposed here is not the only formulation that exists. Nevertheless, the purpose of this master thesis is not about this problematic, so the interested reader is invited to read more by itself in the literature.*

1.3 State-of-the-art

The two previous subsections introduce some mathematical background in graph theory and show how theses notions can be applied to routing problems as the TSP for example. Now that we have theses bases, we will do a tour of the existing problems in the literature in order to find if our specific problem has already been addressed, what solutions are proposed by authors and what are the issues that can arise.

Remark. *Routing problems have been widely studied. Several problems have been defined in the literature, and following list is not exhaustive in this way. We limit here to some problems that seem close to our problematic.*

1.3.1 Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP) already introduced in the previous section, is the problem of finding the shortest hamiltonian closed path through a complete graph where vertices represent cities and edges represent routes between cities with their associated distance as weight. As it is explained by Applegate et al. [2], the origin of this problem is not very clear. Indeed, the first time the term appears on the literature in 1949 in the report *On the Hamiltonian game (a traveling salesman problem)* by Robinson [32] but it seems clear that the term was already in use during the 1930s or the 1940s. This problem and his variants rest widely studied because of his multiple applications and the difficulty of finding an algorithm to obtain the optimal solution for a large set of cities. Indeed, if we take a graph with only 16 cities, there are $\frac{15!}{2} = 653,937,184,000$ distinct routes to explore[1]. A brute-force approach is not possible but the literature proposes different heuristic approaches in order to obtain good solutions in reasonable time. Rego et al. [31] present and compare some algorithms that can be used in order to solve the TSP problem.

[1]We divide per 2 because of the symmetry of the problem : the path $e_1, ..., e_n$ and the reversed path $e_n, ...e_1$ have the same distance.

1.3.2 Traveling salesman problem with profits

The traveling salesman problem with profits (TSP with profits) is a variant of the TSP where a notion of profit is added. A profit (in $ for instance) is assigned to each city and the salesman can collect it if he goes to this city. Now the salesman, which starts from a specific city $v_0 \in V$, is not forced to pass in each city and he wants to maximize his total profit. However, each journey has a cost (proportional to the distance) and this must also be taken into account. In other words, he has two opposite objectives: the first one pushes the salesman to travel in order to collect profits whereas the second one has the opposite impact by encouraging him to minimize the traveled distance.

From this idea, three definition variants exist depending on the way the two objectives are defined [9] :

[i] The two objectives are combined in one objective function.

[ii] The travel cost is considered as a constraint.

[iii] The total profit is considered as a constraint.

Like the TSP, the TSP with profits is defined on a complete undirected graph $G = (V, E)$. Let's now introduce two new notations p_i and y_i associated to each vertex $v_i \in V$. A value p_i corresponds to the profit that can be gained when the salesman passes through city i. A decision variable y_i is associated with each vertex $i \in V$ and defined as follows:

$$y_i = \begin{cases} 1 & \text{if the solution contains the vertex } i \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

The decision variables x_{ij} that was defined above by the equation (1.1) is also kept.

The TSP with profits is then defined on the following constraints as described in [9]:

$$\sum_{j \in V} x_{ij} = y_i \qquad \forall i \in V \tag{1.8}$$

$$\sum_{i \in V} x_{ij} = y_j \qquad \forall j \in V \tag{1.9}$$

$$y_0 = 1 \tag{1.10}$$

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq y_k \qquad \forall S \subseteq V \setminus \{v_0\}, S \neq \emptyset, \forall k \in S \tag{1.11}$$

$$y_i \in \{0, 1\} \qquad \forall i \in V \tag{1.12}$$

The two first constraints described by the equations (1.8) and (1.9) are intrinsic to the definition of the y_i : A vertex is taken ($y_i = 1$) if it has exactly one edge that start from this vertex ($\sum_{j=1}^{n} x_{ij} = 1$) and one edge that arrives to this vertex ($\sum_{i=1}^{n} x_{ij} = 1$). And the vertex is not taken ($y_i = 0$) if there is no edges that start nor arrives to the vertex. The constraint (1.10) specifies that the tour must pass by the starting city v_0. As for the TSP, in order to avoid subtour, subtour elimination constraints described by (1.11) are used. The notation $\delta^+(S)$ is used for this constraint to designate the set of all outgoing arcs from the set of vertices S, that

is to say, all arcs (i, j) for which $i \in S$ and $j \notin S$. The notation $\delta^-(S)$ has the same meaning but for incoming arcs, that is to say, $\delta^-(S)$ is the set of all arcs (i, j) such that $i \notin S$ and $j \in S$.

All three variants [i], [ii] and [iii] share in common the decision variables x_{ij} and y_i and the constraints (1.8) to (1.12).

We now further describe the three different definitions.

Profitable Tour Problem (PTP)

The definition [i] (according to the list above) is called the profitable tour problem (PTP) by Dell'Amico, Maffioli, and Värbrand [7]. The objective function is described formally by

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} - \sum_{i=1}^{n} p_i y_i \tag{1.13}$$

Tour Orienteering Problem (TOP)

The definition [ii] is named the Tour Orienteering Problem (TOP) by Mansini, Pelizzari, and Wolfer [21]. In this case the objective function is

$$\max \sum_{i=1}^{n} p_i y_i \tag{1.14}$$

And the travel cost is limited by the following constraint :

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \leq c_{\max} \tag{1.15}$$

Where $c_{\max}$ is the maximal travel cost allowed.

Prize-collecting TSP (PCTSP)

Finally, the definition [iii] is named the prize-collecting TSP (PCTSP) by Archetti and Speranza [6]. For this formulation, the objective function is the same as for the TSP (see equation 1.2) and the following constraint is added

$$\sum_{i=1}^{n} p_i y_i \geq p_{\min} \tag{1.16}$$

where $p_{\min}$ is the minimal profit that the salesman must collect.

A generalization of the TSP This last formulation can be easily seen as a generalization of the TSP. Indeed, if we take the objective function (1.2) under the constraints described by equations (1.5), (1.8) to (1.10) and (1.16), we can consider the TSP as a particular case of the TSP with profits. This specific case arises when $p_i = 1$ for all vertices and $p_{min} = n$; thus we have the case where $y_i = 1$ for all vertices. And, in this case, the equations (1.8) and (1.9) are reduced to the equations (1.3) and (1.4).

1.3.3 Problems related with the TSP with profits

Prize-collecting Rural Postman Problem (PRPP)

The Prize-collecting Rural Postman Problem (PRPP) – firstly named Privatized Rural Postman Problem by Aráoz, Fernández, and Zoltan [5] – is described by Aráoz, Fernández, and Meza [4] as a variant of the PTP (definition [i] of the TSP with profits) where the profit is not associated to the vertices but to the edges. In addition, each edge can be traversed more than once but the profit is collected only the first time (but the travel cost is counted each time).

Clustered Prize-Collecting Arc Routing Problem (CPCARP) The Clustered prize-collecting arc routing problem (CPCARP) is a variant of the PRPP proposed by Aráoz, Fernández, and Franquesa [3] where the customers are in groups. The profit is earned only if all customers of the group are served. In other words, the profit of a customer is null as long as all the others customers of his group are not served.

Orienteering Problem (OP)

The Tour Orienteering Problem (TOP) mentioned as the [ii] version of the TSP with profits is in fact a particular case of a more general problem called the Orienteering Problem by Vansteenwegen, Souffriau, and Van Oudheusden [38] where the constraint of returning to the starting point is removed. So the problem consists in finding a path between two vertices maximizing the collected profit such that the travel cost does not exceed a given upper bound. We can find this problem under several names in the literature. Laporte and Martello [16] call it the Selective Travelling Salesman Problem (STSP) and Kataoka and Morito [13] call it the Maximum Collection Problem.

Arc Orienteering Problem (AOP)

The Arc Orienteering Problem (AOP) proposed by Souffriau et al. [37] is a variant of the orienteering problem where the graph is orienteed and where the profit is assigned on the arcs instead of on the vertices.

The Most Scenic Path Problem Lu and Shahabi [19] focus on the algorithms that can be used to solve the AOP in the case of a real-world large-scale road network. Their work is about proposing the most scenic path within 300 milliseconds. Indeed, they want propose an interactive mobile and online application that requires fast response time. Their article raises many issues that come when we will work with a large-scale road network. In this case, exact algorithms fail to solve the problem and the common approximate algorithms are too slow. The two algorithms that they point as relevant for this case are the GRASP algorithm described by Souffriau et al. [37] and the ILS algorithm described by Verbeeck, Vansteenwegen, and Aghezzaf [39]. They finally propose improvements of the ILS algorithm by using ellipse pruning and spatial indexing.

Cycle Trip Planning Problem (CTPP) Verbeeck, Vansteenwegen, and Aghezzaf [39] formulate the Cycle Trip Planning Problem (CTPP), a kind of AOP specially designed for bike trips. This formulation will find a closed trail (closed walk without repeated arcs[2]) with the highest profit in a directed graph. The travel cost of the closed trail must be comprised between a given lower bound and an upper bound. Moreover, the starting vertex is taken from a set of possible starting vertices. This last modification is done, for instance, to allow the selection of all parking available in a defined region as a possible starting point of the trip.

Outdoor Activity Tour Suggestion Problem (OATSP)

Maervoet et al. [20] define the outdoor activity tour suggestion problem (OATSP) that consists in finding a tour between lower and upper distance bounds that maximizes the sum of the average arcs attractiveness and the sum of vertices attractiveness. Their definition associates a profit (the attractiveness) to each arc but also to each vertex. Their also allow multiple visits of the same vertex and reverse arc visits but the profit of each arc is taken into account only if the arc is not directly preceded by its reverse arc in order to not encourage U-turns.

1.3.4 Summary and conclusions

In this subsection we have presented routing problems found in the literature that seem to be close to our problematic. Table 1.1 summarizes the main properties of theses problems. In our case, the distance must be near a distance given by the user, that is to say between a lower and a upper bound. Indeed, we cannot suggest to the user a route with a distance bigger than he asks even if this allows seeing more beautiful roads. We cannot neither suggest to the user a route with a smaller distance even if he is in an unpleasant location where there is no pleasant roads at all. Thus, the PTP (and PRPP or CPCARP) is not suitable for our problematic. Similarly, the TSP and the PCTSP are not suitable because they would minimize the distance. In fact, the distance must be a constraint of our problem (see the column *Constraints* in the table to see which existing formulations propose that). Furthermore, we decided to assign profits on roads and not on particular points of interest (POI). This choice is mainly based on the quote: *It's not the destination but the journey that matters*. We indeed focus on the route and not on particular POI that are often touristic. So problems that propose only to assign a profit on vertices are not suitable either.

Based on theses restrictions, we can see that the AOP, his variant the CTPP and the OATSP are very close to our problematic.

Based on the mathematical background and the literature review that we have presented in this chapter, we can now define more precisely our problem. The next section is dedicated to the definition of our problem in mathematical terms and the argumentation of the choice made.

1.4 Definition of our problem

Now that we have seen the different problems presented in the literature and their formulations we will define our particular case. We are interested in finding a tour that start from a given

[2]To prevent U-turns, their consider that the trail cannot contains both arc and the corresponding reverse arc.

Name	**Objective**	**Profit on**	**# visits allowed**	**Constraints**	**Note**
TSP	$\min D$	-	$V : 1$	hamiltonian path	
TSP with profit		V			
$\hookrightarrow$ **PTP**	$\max(P - D)$	V	$V : 0 \to 1$		
$\hookrightarrow$ **TOP**	$\max P$	V	$V : 0 \to 1$	$D < D_{max}$	(Variant of OP)
$\hookrightarrow$ **PCTSP**	$\min D$	V	$V : 0 \to 1$	$P > P_{min}$	
PRPP	$\max(P - D)$	E	$E : 0 \to \infty$		(Variant of PTP)
$\hookrightarrow$ **CPCARP**	$\max(P - D)$	E	$E : 0 \to \infty$	cross all or none of the edges of a cluster	
OP	$\max P$	V	$V : 0 \to 1$	$D < D_{max}$	
AOP	$\max P$	A	$A : 0 \to 1$	$D < D_{max}$	
$\hookrightarrow$ **CTPP**	$\max P$	A	$A : 0 \to 1$	$D_{min} < D < D_{max}$	Starting V from a set of vertices
OATSP	$\max P$	A, V	$A : 0 \to 1$	$D_{min} \leq D \leq D_{max}$	

Legend: D : Distance, P : Profit, V : Vertex, E : Edge, A : Arc (oriented version of edge)

Table 1.1: Summary of the state-of-the-art for the routing problem

vertex $v_0 \in V$ in our graph that maximize a total value given a constraint over the distance.

Let's first define our graph. We work with a graph representing the road network. Therefore, all crossroads are vertices and all road segments are edges linking two vertices. Because we can have multiple road segments that link together the same two vertices and since we don't have a way to select a priori one or another[3], our graph is not a simple graph. We can have multiple edges with the same two vertices as extremities and loops in our graph. The road network for slow mode of transportation is usually undirected but we decide to work with a directed version of the graph. Each road segment is represented by two directed edges (arcs) in the both directions. This representation has the advantage of making easier the formulation of our problem and has especially the advantage to be general. We can therefore improve our application in a later stage for instance by taking into account the one-way cycleways.

Our directed graph composed by a set V of n vertices, a set A of m arcs and a relation φ that links each arc $a \in A$ to his two extremities, the departure vertex $v_{\text{start}} \in V$ and the arrival vertex $v_{\text{end}} \in V$. These two extremities can be the same, and two arcs can have the same two extremities.

Each arc $a \in A$ is also linked to a length l_a (the length of the road segment) and a profit value p_a (the pleasant value of the road segment) and a road segment identifier $\#(a)$ that is the same for the two directions of the same road segment. The length is always positive but the profit value can be positive or negative. We consider also that the profit value is independent of the direction:

$$p_i = p_j \qquad \forall i, j \in A \text{ s.t. } \#(i) = \#(j) \tag{1.17}$$

To formulate more easily our problem and our constraints, we add decision variables. The first one, x_a, is about the presence of each arc a in the solution and the second one, y_v is about

[3] A naive approach is to take the one that has the greatest profit, but given the constraint on the distance this is not always the best choice.

the presence of each vertex in the solution.

$$x_a = \begin{cases} 1 & \text{if the tour contains the arc } a \in A \\ 0 & \text{otherwise} \end{cases} \tag{1.18}$$

$$y_v = \begin{cases} 1 & \text{if the tour passes through the vertex } v \in V \\ 0 & \text{otherwise} \end{cases} \tag{1.19}$$

These two decision variables are clearly linked together as we will see in the constraints of our problem.

As mentioned before, the mathematical notation $\delta^+(S)$ corresponds to the set of all arcs that quit S, that is to say, arcs for which the starting vertex is inside S and the arrival vertex is outside S. The notation $\delta^-(s)$ is, for its part, the set of all arcs that come in S.

Our purpose in this chapter is to find a closed trail starting from a given vertex $v_0 \in V$ that maximizes the sum of the profit of each arc in the walk under a constraint of distance. We do not want to restrict to not pass twice through the same road segment (in one direction then in another) because the map can contain some critical roads, for instance if we have only one access to a beautiful forest. Nevertheless, we don't want a tour where we take a lot the two directions of a same road segment. To deal with this problem, we decide to count only once the positive profit associated to each road segment. With all this in mind, we can now formulate mathematically our problem as following:

$$\max_x f(x) \tag{1.20}$$

$$\sum_{a \in \delta^-(v)} x_a \leq M y_v \qquad \forall v \in V \tag{1.21}$$

$$\sum_{a \in \delta^-(v)} x_a \geq y_v \qquad \forall v \in V \tag{1.22}$$

$$\sum_{a \in A} l_a x_a \leq D_{\max} \tag{1.23}$$

$$\sum_{a \in A} l_a x_a \geq D_{\min} \tag{1.24}$$

$$\sum_{a \in \delta^+(v_0)} x_a \geq 1 \tag{1.25}$$

$$\sum_{a \in \delta^-(v)} x_a = \sum_{a \in \delta^+(v)} x_a \qquad \forall v \in V \tag{1.26}$$

$$\sum_{a \in \delta^+(S)} x_a \geq y_k \qquad \forall S \subseteq V \setminus \{v_0\}, k \in S \tag{1.27}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{1.28}$$

$$x_a \in \{0, 1\}, \qquad \forall a \in A \tag{1.29}$$

Where the equation (1.20) describes the objective that we define more in detail in the next paragraph. Constraints 1.21 and 1.22 link the decision variables together by using a M constant that is large enough, that is for instance $M \geq |A|$. Constraints 1.23 and 1.24 force the total distance of the tour to be between the desired minimum distance, denoted by $D_{\min}$ and the

maximal distance, denoted by $D_{\max}$. Since we search for a tour that starts from a given vertex v_0, we impose then in the constraint 1.25 that at least one arc leaves this vertex. Constraints 1.26 impose the conservation of the flows, i.e. the number of arcs that enter in a vertex is the same that the number of arcs that leave this vertex. Constraints 1.27 are the *subtour elimination constraints* to avoid having multiple independent subtours in our solution. Finally, constraints 1.28 and 1.29 indicate simply the domain of the decision variables.

Now that we have this formulation, we can precise our objective function, noted $f(x)$ in equation (1.20). Because we want to take into account the profit associated to a road segment only once (and not twice when the road segment is taken in its two directions); we add the b_{ij} notation as

$$b_{ij} = \begin{cases} 1 & \text{if } \#(i) = \#(j) \\ 0 & \text{otherwise} \end{cases} \tag{1.30}$$

With this notation, it is now possible to define our objective function $f(x)$ as follows:

$$f(x) = \sum_{a \in A} x_a p_a - \sum_{\substack{i \in A, j \in A \\ i \neq j}} \frac{\max(0, x_i x_j b_{ij} p_i)}{2} \tag{1.31}$$

The first part of the objective sums the profit of all arcs that are taken in the solution, whereas the second part removes from this count the positive profit of arcs that are taken in the both directions. We take the maximum between the profit and 0 in order to keep the negative impact of negatives profits in the two directions. Then, we divide by 2 because, when we pass twice by the same road segment, we want to remove once the profit of this road segment but b_{ij} and b_{ji} are both equals to 1. Note that by the definition (1.17) the profit p_i is equivalent to the profit p_j.

This definition of the objective can suffer from some drawback if the profit value is always positive. Indeed, in this case, the algorithm would increase the total distance in order to increase the objective even by taking road segments that are not really pleasant. For this reason, we keep in mind that the profit value defined on the second chapter must contain negatives values for road segments that are not pleasant at all and a positive one for roads that are pleasant.

1.5 Conclusion

This chapter was about the problem of routing optimization. In its first part we presented some important definitions of graph theory in order to get familiar with it and to better understand routing problems. We then introduced the routing problem in itself by presenting some routing problems from the literature. The purpose of this state-of-the-art was to get familiar with routing problems but also to see if our particular case has already been formulated and studied. However, we did not found any formulation that corresponds exactly to our problematic. That is why we proposed in the last section of this chapter, our own mathematical formulation of the problem to solve. During all this chapter we assumed that value representing the pleasure of the road is available for each road segment. In the next chapter we will discover how we can compute this value. Finally, the Section 3.5 presents how we have implemented and solved the routing optimization problem in our final application.

Chapter 2

Second problem : Pleasure related to a route

The first chapter is devoted to the routing optimization problem. The purpose is to find a tour that maximizes the total *profit value* of the route and that has a distance comprised between a given lower and a upper bound. As the profit value must in fact reflect the pleasure felt by the user while travelling the route, we refer to it as the pleasant value. The problem analysed in this second chapter is thus the problem of assessing the *pleasant value* of a particular section of road and more generally the pleasure associated to an entire route, that is, to a particular solution.

This concept of pleasure obviously depends on multiple factors, some of them being different from a person to another one. Indeed, several aspects may be integrated through the concept of the pleasure of a road : the quality of the road, the beauty of the scenery, the sound environment, etc.

This chapter is introduced in section 2.1 by a more precise specification on what we call a *pleasant route*. Then, section 2.2 presents a short review of the associated literature. Section 2.3 presents which data can easily be used for our purposes. We also discuss the relevance of theses data. Finally, based on the conclusions of the first three sections, section 2.4.3 defines how we compute our *pleasant value* of a route, namely how pleasant it is for a customer to travel along it.

2.1 Introduction : Problem specifications

The pleasure related to a route is a subjective but also broad concept. It is thus important to restrict it and to describe what we consider as a *pleasant route* in this work. An intuitive idea is to said that a *pleasant route* is a route that is "*emotionally pleasant*" [29]. This can be also clarified by saying that a *pleasant route* is a route that propose a good compromise between different criteria (to be defined later), like the quality of the road, the visible scenery, the sound environment, the attendance of it, etc. Namely, any measurable aspect that could improve the experience of the user while he travels. The purpose of our work is finally to propose to the final user a route that makes him happier than a sufficiently lage majority of other possibles routes.

From these ideas, we can see that we are talking about users but who will they be ? What is our target audience? Final users that we target are not only tourists in an unknown location, but also residents willing to discover new pleasant tours nearby their house. For instance, this can be very useful for runners and cyclists who want to do a tour of a precise distance starting from their house. Final users that we will target can also use different mode of transportation, and we want to mainly focus on *soft* transportation modes: cyclists, walkers, hikers, runners, etc. Indeed, fast transportation modes don't allow people to fully enjoy the beauty and the quietness of a road and are also already more studied.

As the pleasure depends on the mode of transportation and on others personal preferences (e.g. cyclists generally want to avoid cobblestone paving), we want propose to users to give their preferences (through a user friendly interface) before the computation of the best route for them. We also want our system to be available in the most countries as possible and in particular in Europe, including areas that are not especially touristic nor highly visited so that we do not neglect certain persons or destinations.

2.2 State of the art

Although the assessment of the beauty of a route are a very recent research area, the literature already contains some interesting studies. Firstly, it can be interesting to observe which attributes influence the scenic value of a route. Alivand, Hochmair, and Srinivasan studied this influence in they article "Analyzing how travelers choose scenic routes using route choice models" [1]. Attributes they have taken into account can be separated in two categories :

[i] Attributes related to the route in itself : road type, shape, travel time, distance,...

[ii] Attributes related to the surroundings of the route : proximity of mountains, water bodies, green areas, or historical buildings.

They have identified which attributes have an influence on scenic route selection by comparing attributes on known scenic routes with alternatives routes (the fastest routes). For the first category of attributes ([i] in the list above), results show that the scenic routes use more secondary and local roads and less primary roads than the fastest routes. It is also shown that curvier routes are more scenic than straight routes (curviness attribute is computed by the method proposed by Navratil [22]). And the number of turns (to left and to right) are greater in scenic routes. For the second category of attributes, significant results show that forests, mountains, parks/gardens and water bodies are in a greater number in scenic route while urban and cultivated area have a negative impact on the scenic perception of the route. In the second category ([ii] in the list above), they have also compared the *Panoramio.com* [25] photo density, which is the number of pictures that are taken along each road. This density is a little greater on scenic routes than on fastest routes.

The literature therefore shows that there are multiple attributes that can be taken into account in order to suggest scenic routes. In the remainder of this section, we do a tour of what has been implemented in the literature and what are the conclusions that we can draw.

2.2.1 Visibility of scenic sights

We have seen that the surroundings of the route can influence its scenic value. In their work, Zhang, Kawasaki, and Kawai propose a *Tourist Route Search System* that searches a beautiful route that links several tourist spots. Their recommendations are for people that want travel by car and enjoy beautiful scenery while driving. They define *scenic sights* as *"landscapes that can be enjoyed as a distance view", such as Mt. Fuji, Eiffel Tower, Tian'an Gate, and so on.* Their work is decomposed in two phases :

(i) They propose to the user a selection of tourist spots and scenic sights extracted from an internet search.

(ii) They compute a route between tourist spots that passes through roads from which you have a view on scenic sights.

The most interesting part for us is thus the second phase. To check the visibility of scenic sights, they segment the road into n points spaced from each others by the same distance. Then they compute for each point if the scenic sight is visible or not. To do that, they construct a 3D model of the map by using a Digital Elevation Model (DEM). The visibility value v_i is set to 1 if the scenic sight is visible from the node i, and to 0 otherwise. Finally they compute a *visibility rate* s as

$$s = \frac{1}{n}\sum_{i=1}^{n} v_i$$

To take into account the fact that a scenic sight can be a large region (e.g. mountain), they use multiple scenic points over the region to define the scenic sight. Assuming a set of m scenic points, they define the *total visibility rate* S as

$$S = \frac{1}{n}\sum_{i=1}^{n}\{\bigvee_{j=1}^{m} v_{ij}\}$$

This is an interesting approach in the fact that they look at the view that we can have from the road. The problem is that they focus on scenic sights, although in some location and for some pleasant walk this is not the only variable. If you like to walk in the forest, then it is for instance not a good approach. Another problematic is that the DEM used does not contain elevation of buildings whereas buildings can block the view.

2.2.2 By using informations shared on internet as sensors

Since the Web 2.0 appeared, people are more and more invited to share their experience on internet. As we shall see, these information can be used as sensors to determine location that are interesting to visit, which roads are more beautiful or have more scenery view than others. For instance, Kurashima, Tezuka, and Tanaka [15] and Kori et al. [14] study tour generation systems based on information extracted from blogs. This approach requires a difficult extraction of information from blog and is thus not well studied. Contrariwise, the usage of geo-tagged photos shared on internet is an approach that is rather widely used by many authors in the literature.

Geo-tagged photos

Websites like *Flickr.com* [10] or *Panoramio.com* [25] allow people to share their photographs on internet and this constitutes a large database of geo-tagged photos. Theses geo-tagged photos can thus be associated to a location and displayed on a map[1]. As observed by Lu et al. [18], Hochmair [12], Zheng et al. [43], Alivand, Hochmair, and Srinivasan [1], and Skoumas et al. [36], the density of theses pictures can be used to approximate the interest of a road. For instance, Zheng et al. [43] use this approach for their GPSView application, *A scenic Driving route planner*. They base their system on this premise: *if a large number of photos are densely distributed along a roadway, then this roadway is a scenic one with a series of scenic sight spots, or Points-Of-Interest (POIs), on the roadside.* They propose the usage of this source of data after having analyzed the following difficulties. First, there are no database that provides a large and reliable list of scenic roadways. Secondly, it is difficult to estimate the scenic and aesthetics qualities of a roadway as it depends on the sightseeing experience of the traveller. Thirdly, to enjoy a scenic sight, scenic spots must be visible from the roadways. Finally, the algorithm that computes the route must be effective in real time thus this is not possible to make too complex computations.

Advantage of this approach is that they can propose to the final users a route that has a scenic view that is commonly appreciated by others people. Nevertheless, this approach seems to propose only tourist roads and can suffer from the lake of photos in certain region of the world (less than 10 photos in the municipality of Ottignies-Louvain-la-Neuve in Flickr, and less than a thousand on panoramio[2]). We can also ask the question of the relevance of the photography since people are taking more photos of their everyday life not especially related to the beauty of the location therefore a good filtering of the photography must be done before the usage of this source.

Users recommendations

With the Web 2.0, another approach to find a pretty tour is the use of tours that other people have done before and shared through websites like *OpenRunner.com* [23] or *WikiLoc.com* [41]. This kind of database allows everyone to upload the trail of their walk. Generally, the trails are categorised by sport (hiking, running, cycling, horse riding, etc.) and users can add comments and ratings. The advantage of this solution is that tours are based on the real experience of the users that have done the walk before. Nevertheless, their are some drawbacks to raise. First, if nobody has shared a walk near you, you don't have any mean to obtain a pleasant walk. And even if there are some tour near you, it is possible that the distance does not correspond to your expectations. There is actually little chance to find a tour that starts from your location and that has the desired distance. Secondly, if we want to take into account your preferences (like: I prefer forests than fields) we have generally to read the precise description or analyse the tour on a map for each tour.

[1] See `https://www.flickr.com/map` and `http://www.panoramio.com/map`
[2] in February 2016

2.2.3 By using Google Street View or other street photographies

In section 2.2.2, we have seen some manners of how to use data shared through internet by users as sensors. In this section, we now show usages that we can make from street photographies. The first approach by Quercia, Schifanella, and Aiello [29] is in the boundary with the previous section since it use also information shared by users on their website to assess the quality of the road.

The Shortest Path to Happiness

We have seen that the information shared by people can be useful for evaluate if a route is scenic or not. Quercia, Schifanella, and Aiello [29] use another approach to find what they call *the shortest path to happiness*. Their approach evaluates three characteristics of road: the beauty, the quietness, and the happiness. To evaluate theses qualities, they propose a simple game through a website: `UrbanGems.org`. At each round, two pictures extracted from Google Street View or from Geograph[3] are presented and the user is invited to respond to the question *Which place do you find more beautiful?* (or *more quiet* or *happier* in the two other gaming modes). Based on the result of this game, they are finally able to assign a score to each street and are thus able to give alternatives to the shortest path that are more beautiful, more quiet or happier. The system that they propose find paths that improve between 26% and 30% the value optimized (beauty, quietness or happiness) in comparison with the shortest path. The recommended path is on average 12% longer. But the more interesting measure that they provide for their system is the actual perception of the route proposed by users. Results show that alternatives paths are actually more beautiful than the shortest path. For the quiet path, the quietness is also better and we can see the same conclusion for the happiness. They also point an important affiliation between beauty and happiness of routes.

They finally use the flickr geo-tagged pictures in order to compare the resulting path to other paths they have proposed before (beautiful, more quiet and happier). In order to sort the many photographies they use tags associated with each Flickr photography and the dictionary called "Linguistic Inquiry Word Count" (LIWC) in order to assign to each tag a category of words. Categories obtained from this dictionary allow them to sort between positives and negatives pictures. The path is then rated by 30 interviewed persons and results show a median of 2 (on a scale of 0 to 4); that corresponds to a neutral answer (not negatively nor positively). However, by comparing the person that have rated below 2 and above 2, they observe that the Flickr variant is preferred by women and short-term residents (of London, where the study is done). At the end, they discuss some observations and questions. Two of them seem interesting to take into consideration for our own work:

(i) **Personalization**: They raise that *perceptions of urban qualities such as happiness might well differ from one individual to another*. It is thus interesting to propose to users enough settings in order to personalize their experience.

(ii) **Limited Contextual Representation**: Some feedbacks given by participants indicate that weather and time are factors that can play a role. Indeed, some places are nicer during the weekend than during the week at busier times. The beauty can also vary with seasons or depending on weather conditions.

[3] `http://www.geograph.org.uk`

No more Autobahn! Scenic Route Generation Using Googles Street View

At the time of writing theses lines, Runge et al. [34] are currently publishing a new a paper that will be presented during the IUI meeting in March 2016. Their work also uses Google Street View as source to provide scenic routes. But they propose an automatic system. They use this source of data because they judge than a 3D model is computationally too expensive and do not take into account details like presence of flora, trees, etc. The system that they propose retrieve pictures of roads from Google Street View and analyse them in order to classify into the following categories: *Nature and Woods*, *Water*, *Sightseeing*, *Fields*, *Mountain* and *Non-scenic*. The system use a training set of 111 scenes to learn and predict the probability of each categories for a new street picture. They finally divide the map in a grid and try to assign to each cell of the grid a category. For a user that likes *Nature and Woods* view, the routing algorithm finally proposes the route that maximizes the cells classified as *Nature and Woods* in the given time.

2.2.4 Other approaches and conclusions

Hence, there can be many expectations about a route and there are also a lot of different approaches. Many authors presented in this literature review talk about the scenic value of a road, but we have also seen other criterion like the quietness and happiness of a route. Approaches vary when we design an application especially for a city or for a type of user. The pleasure related to a route can also vary if we want to be alone or not. For instance, Posti, Schöning, and Häkkilä [28] propose an asocial approach that wants to avoid all contacts with other people. Tourist spots are in this way not always a good choice and thus geo-tagged pictures are not always appropriate. The usage of Google Street View pictures can be a solution but this limits the routing to road that has been photographed by Google thus pedestrian paths are rarely taken into account. The usage of database of paths shared by other users (like in *OpenRunner.com* [23] or in *WikiLoc.com* [41]) will not allow either to provide to users a personalized experience and tour within their time budget and their initial location.

It is thus important to construct a solution that takes in input the preference of the user and is able to provide tour that fits theses preferences. We have also to take into account the computational difficulty of the approaches since our application must respond within a short time. In this way the approach proposed by Zheng et al. [43] is not adaptable to our situation since the computation of a 3D model seems computationally too expensive.

2.3 Available sources and data

The beginning of this chapter shows that the pleasant value of a road is a difficult value to assess and that multiple parameter can influence this value. We choose now which parameters are the most relevant according to the data at our disposal. For this purpose, we now examine which source of data are freely accessible and what is the nature of the data that we can access. Only after that, we can determine how we will perform our approximation of the pleasant value.

2.3.1 Geotagged photography

As explained on the previous section, one source of data can be geotagged photographies shared by people on internet. For instance, *Flickr.com* [10] propose an API that allow us to retrieve geotagged photographies with latitude and longitude. *Panoramio.com* [25] propose also a similar API. The problems of this kind of source is that there is a risk of having a lot of data for tourist spot but less for more quiet and less tourist region. Another problem is that there are photographies of party, of attraction park, etc. thus a filter must be applied before the using of photographies and this reduce the number of available data.

2.3.2 Maps

The more intuitive and common way to have information about a location or a path is to see it on a map. This kind of representation is used from long time ago and is currently the easier way to represent on paper (or on screen) the topography of a location. With the growing of internet, maps are not anymore a static representation but can be interactively visualised on internet through different web services or softwares. For instance you can easily interact with an online map by zooming on a particular location and seeing more and more details. You start from a map that show a country with its main highways and towns then you can zoom to see a particular street with the details of buildings. The available maps can be separated in two categories : Commercials maps and Collaborative maps[4].

Commercial maps

The most known interactive map on internet is surely *Google Maps*[5]. This map presents a lot of data and can be accessed through their website but also through multiple APIs[6]. Theses APIs allow conversion between addresses and geographic coordinates (Geocoding), getting the elevation data of any point on the world, or even knowing the speed limitation of a specific segment of road. It is also possible to get the routing instruction between two locations. This is thus a very diversified source of data. A disadvantage to point for this source is that APIs are limited in terms of number of requests, except if you pay. APIs are also limited in terms of raw data that are accessible: it is possible to compute a route between two location using their algorithms, but it is not possible to have the graph of all roads in an area. In addition, it is not possible to get precise characteristics on the road type (except the speed limit in the paid version of the API).

They exist other similar APIs like *Bing Map*[7] from Microsoft, but it seems that they suffer from similar issues.

Street View The approach of Runge et al. [34] mentioned in the previous section use Street View images from the *Google Street View API*[8] but this seems not a suitable solution for pedes-

[4] In fact, theses categories aren't clearly distinct because some commercial maps uses theirs users to improve their maps in a collaborative way.

[5] `https://maps.google.com/`

[6] Complete presentation of theses API on `https://developers.google.com/maps/web-services/`

[7] `https://www.bing.com/maps/`

[8] `https://developers.google.com/maps/documentation/streetview/`

trian or cycling routing since the number of photographies of pedestrian and cycling paths is not representative enough.

Satellite imagery In online commercial maps, it is frequent to have a satellite view. This view gives a certain kind of raw information that can be analysed and transformed in more precise information. For instance, it is imaginable to distinguish natural area from industrial area by the colour more green or more grey of the aerial photography. Google allow retrieving theses aerial photographies for any locations in the world with different zoom levels through their *Google Static Maps API*[9]. Informations that we can retrieve in this way can be a good element to determine the type of neighbour of a road but requires a complex conversion in order to be really exploitable to our problem, which is not in our scope.

Collaborative map

Another source of geographic data is information constructed by volunteer in the same principle of the famous website Wikipedia but adapted for the creation of a map. This phenomena is called *Volunteered geographic information* (VGI) [11]. There exist some websites that propose this like Wikimapia[10], OpenStreetMap[11], or also Geo-Wiki[12]. OpenStreetMap (OSM) seems the most relevant of it for our purpose since it is currently used by many services and softwares[13] and the OSM community seems to be really active.

The principle of OSM is simple: all people can modify, refine and add geographic data on the map. All these data are publicly available under the *ODC Open Database Licence (ODbL)*[14]. Therefore, an interesting point derived from this model is that all people on the world can access to the raw data of the map in order to construct application based on geographic information.

For our work, we can thus access freely on the whole raw data of OSM. But what kind of information are in fact accessible on this map ? As explained, data available are data that have been added by volunteer. Theses data are divided in three kind of elements: *Nodes*, *ways* and *relations*.

- A **node** is a specific location on the map (with a single latitude and longitude).
- A **way** is a polyline, that is a *walk* if we reuse the definition 8 (p. 6) of the previous chapter, thus ways are used to represent linear things like roads but also to describe area boundaries (*closed walk*). It is thus composed on multiple nodes. The figure 2.1 shows an example of ways in OSM.
- A **relation** is a data structure that is used to link together elements (nodes, ways and/or other relations). For instance this is used to link roads segments (ways) to indicate a cycling route (like a *Ravel*). This is also used to represent area with holes inside. The figure 2.2 shows a lake with an island that is represented by a relation with two kinds of

[9] `https://developers.google.com/maps/documentation/static-maps/`
[10] `http://www.wikimapia.org`
[11] `http://www.openstreetmap.org`
[12] `http://www.geo-wiki.org/`
[13] The list of service that use OSM can be consulted on `http://wiki.openstreetmap.org/wiki/List_of_OSM-based_services`
[14] `http://opendatacommons.org/licenses/odbl/`

ways: Ways that make the outer boundary (thus, the boundary of the lake) and ways that make the inner boundaries (thus, the boundary of the island).

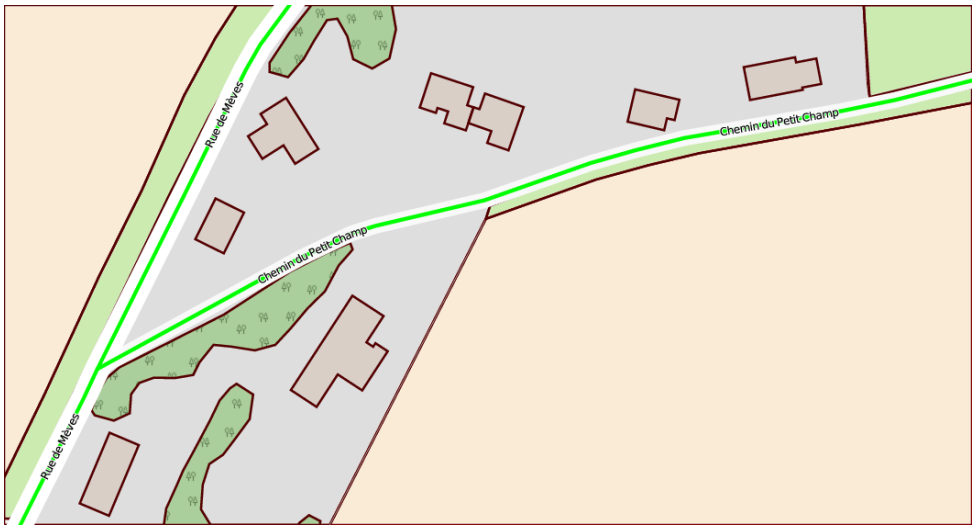


Figure 2.1: Example of ways elements in OSM. Roads are ways (green lines) but (closed) ways can also represent polygons (red outline)

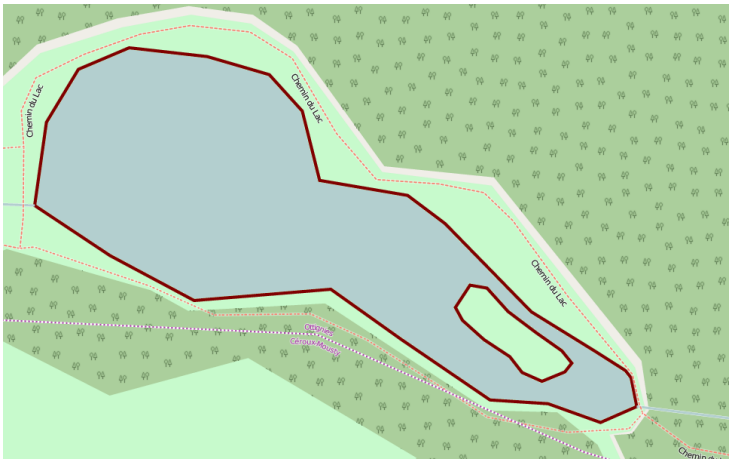
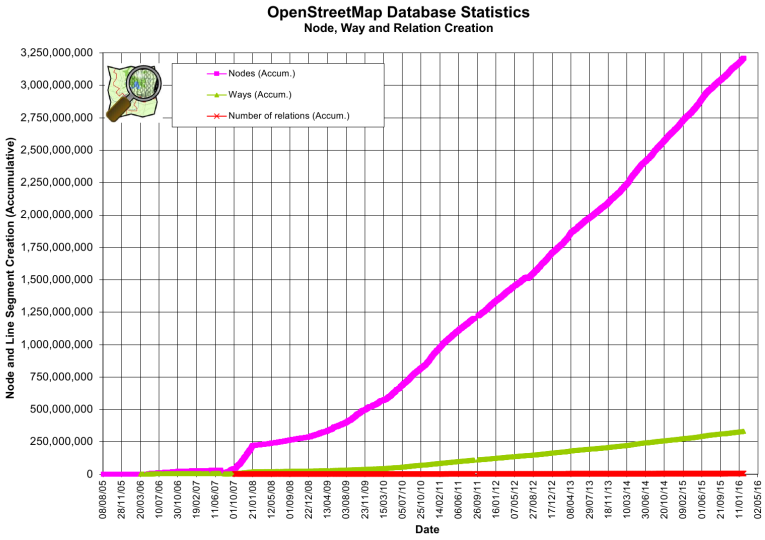


Figure 2.2: Example of relation representing the water part of a lake without the island in order to form a multipolygon

Each element has a unique identifier but can also be associated to one or more *tags*. Each tag is composed by a *key* and a *value* that describe the meaning of the element. For instance, a road can have the following tags: `highway=residential`, `lanes=1`, `maxspeed=50`, `name="Rue du Vieux Chemin de Genappe"`, `surface=asphalt`, `cycleway=track`. This indicate that the street *Rue du Vieux Chemin de Genappe* is a residential road with one lane where the speed is limited to 50 km/h and has a separated lane for bicycle. The meaning associated with the pair key/value is generally easy to understand but is also usually described in the OSM wiki[15]. In reality, keys and values that can be added in OSM are not restricted (i.e. this is a *free tagging system*). This allow everyone to propose and create new tags (and values associated with it). Nevertheless, the OSM community agrees on a certain number of key and value combinations that are proposed and then documented on the OSM wiki. This system has the advantage of being very flexible and adaptive because each real world element can be represented.

[15] `http://wiki.openstreetmap.org/`

We have seen that OSM contains many data: All elements of the real world can be represented very precisely thanks to the free tagging system. Nevertheless, this possibility of representing all the real world is not a guarantee that it is done in practice. So we can ask ourselves whether the OSM data are sufficiently complete and accurate and which tags are actually used. OpenStreetMap was launched in july 2004. Since the creation of the map, we can see that the map tends to be more and more accurate since the topography of the world don't change quickly but contributors continue to add nodes, ways and relations and to refine the existing ones. Figure 2.3 shows that the maps is actively improved each days. The number of relations is very small regarding to nodes and ways but is anyway important since they are currently more than 4 101 400 relations. Recall also that a way is not just an edge between two nodes but a way is a polyline composed by several nodes. Therefore, the fact that the number of nodes increase faster than the number of ways can be explained by the refinement of the shape of existing ways (a way that contains more nodes is more precisely described). We can also note improvements on available editors that make the modification of the map easier for everyone. The development of *Quality Assurance tools*[16] to detect bugs and improve the maps have also surely a positive impact on the relevance of this map.



Source : `http://wiki.openstreetmap.org/wiki/Stats`

Figure 2.3: Evolution of the number of nodes, ways and relations in OSM.

2.4 Value definition

Now that we have seen which data are easily available and usable, we define a precise *pleasant value* for a given tour. There are several ways to define a value over a tour. This value can be a function of the properties of each road segment taken separately, of the neighbourhood of each road segment or also function of the global tour. An example of attributes over the whole route is the number of instructions needed to describe it as been studied by Duckham and Kulik [8].

In our work, we decide to limit ourself to a value that can be calculated for each road segment separately. This seems a relevant choice regarding to our definition of a pleasant tour. This is

[16]See a detailed list on `http://wiki.openstreetmap.org/wiki/Quality_assurance`

also easier to use as we have decomposed the problem of routing and the problem of the pleasant value by assuming that we can compute a value for each road segment.

In this section, we define a value for each road segment. For each road segment i we define a value P_i. A first challenge here is that this pleasant value must be customizable. Indeed, we have seen that it depend on the transportation mode and also on the user. We start by a short description of the data we use based on the analysis done in the previous section. Then we describe what we call a *user profile*. The user profile allow us to tackle the first challenge. Finally, we define precisely the pleasant value for each road segment.

2.4.1 Data

After having analyzed the different source of data in the section 2.3, we decide to base the computation of the pleasant value on the data available from OpenStreetMap (OSM). In these data we are particularly interested in the following elements:

- Ways that represent roads (green lines in figure 2.1)
- Ways that represent area boundaries, i.e. polygons (red outlines in figure 2.1)
- Relations that are multipolygons (polygons with hole(s), figure 2.2)

Ways elements being described by a list of nodes, we are also interested in the nodes. Indeed, each node contains its geographic coordinates. Multipolygon's relations are, for their part, composed of ways. For all these elements we would like know their properties. Theses properties are stored in a list of tags, that is to say, a list of keys associated with values. For instance we can have a tag with the key `highway` and the value `residential` associated to a way. This indicates simply that the way is a residential road.

A challenge with these data is that OSM is not already fully complete and thus it is better that a lack of information on a particular road segment does not affect too badly (or too goodly) the results of our estimation. We have thus to keep it in mind when we define the pleasant value.

2.4.2 User profile

In order to take into account the preferences of the user in the computation of our pleasant value, we introduce what we call a (user) *profile*. This profile contains information about what is most pleasant and what is less pleasant. Since OSM provides useful information through a system of tags, we use theses tags keys and values to describe our profile. We propose to assign to some tags a *preference value* between 0 and 100. A value of 100 indicates that the tag has a very positive impact on the road, a value of 50 indicates that it has a neutral impact and a lower value indicates that it has a negative impact.

We distinguish two kinds of tags in OSM data :

- Tags associated with ways representing roads. Theses tags describes the nature and the properties of the roads in itself.

- Tags associated with the polygons (closed ways) and the multipolygons (relations) that represents areas. Theses tags describes, among others, the natural landcover or the land-use of the area.

Based on this distinction, we divide our profile in two main parts. The next section presents how these two parts are used differently in the computation of the pleasant value. Whereas the road can be described by a list of tags, this is not the case of the environment. Indeed, a road segment can have an environment containing several kinds of elements like forests, rivers, buildings,... For each polygons, we want to assign a pleasant value based on the polygon's tags, on the profile and on the proportion of the polygon near the road.

Preferences values for roads

Listing 1 illustrates how preferences values can be defined for roads in the profile. For instance, the left part of the listing indicates that we prefer tracks, pedestrian and paths over larger roads. We can also note that we add a `default_score` to indicate which pleasant value take when no data are available or when the tag value does not fit any values defined in the profile. Here, the value of 60.0 indicates that if the tag value is no present or is not one of the possible values defined, it is more likely that the road is not a large road but a little one difficult to identify. This allows us to deal with the problem of the incompleteness of the OSM data. The right part of the listings indicates preferences over the surface of the road, that is to say the kind of road pavement.

Based on this definition of the profile, we can define scores for as many tags as we want. To find which tags are widely used (and thus relevant to associate with scores) we can check the *taginfo* tool provided by OSM at the address `http://taginfo.openstreetmap.org/`. This tool lists the tags and allows sorting keys by percentages of use. It also shows which values are possible for each tag key and gives many useful statistics about tags.

Preferences values for road's environment

Listing 2 shows how we define preferences values for polygons that are in the environment of a road. To assign a pleasant score to a polygon, we take the first score in the list that match the tags of the polygon. The special value `*` is used as wildcard to mean any value (excepted the value `no`). We assign finally a default value for the part of the environment that match nothing in our profile or for the part of the map that is poorly mapped.

Listing 3 shows the composition of the final profile. The `road_scores` and `env_scores` parts are presented respectively in listings 1 and 2. To complete the profile, we add two other fields. The parameter $\alpha in [0, 1]$ indicating if we want to take more into account the road in itself or the surrounding of the road and a list of forbidden roads (for instance, a cyclist never want to take a road with steps).

```
{
  "key": "highway",
  "scores": {
    "secondary": 40.0,
    "secondary_link": 45.0,
        "tertiary": 50.0,
        "tertiary_link": 60.0,
        "residential": 70.0,
        "path": 80.0,
        "track": 90.0,
        "pedestrian": 85.0,
        "footway": 70.0,
        "living_street": 80.0,
        "unclassified": 90.0,
        "steps": 70.0
      },
      "default_score": 60.0
}
```

```
{
  "key": "surface",
  "scores": {
    "unpaved": 70.0,
    "cobblestone": 60.0,
    "sett": 65.0,
    "concrete": 60.0,
    "paving_stones": 70.0,
    "compacted": 80.0,
    "dirt": 75.0,
    "earth": 75.0,
    "grass": 90.0,
    "gravel_turf": 70.0,
    "fine_gravel": 70.0,
    "ground": 70.0,
    "mud": 70.0,
    "pebblestone": 70.0,
    "sand": 70.0,
    "paved": 40.0,
    "asphalt": 40.0
  },
  "default_score": 50.0
}
```

Listing 1: Example of profile characterising the value of a road.

2.4.3 Value computation

Now that a user profile is defined, we want to compute a value that assesses the pleasure of the user for each road segment. We propose the following definition of the pleasant value P_i associated to a road segment i.

$$P_i = l_i(\underbrace{(1-\alpha)R_i + \alpha E_i}_{\text{score} \in [0,100] \text{ in } [m^{-1}]} -50) \tag{2.1}$$

We divided our computation in two main terms: the R_i part corresponds to the score assigned to the road in itself whereas the E_i part represents the score associated to the environment of the road segment. The parameter α (between 0 and 1) allows us to decide the relative importance between this two terms. Moreover, in order to have a normalized score, we impose that each of theses terms are between 0 and 100. Thus, the score per meter is also always comprised between 0 and 100. Our routing optimization problem defined in the section 1.4 needs a profit value that is positive for the pleasant road segments and negative for road segments that are not pleasant thus we remove 50 from the score to obtain a value between -50 and 50. Moreover, we multiply by the length of the segment because the score of big road segments must influence more the solution than score of small segments. This definition can then be used for the whole tour in the objective function defined by the equation (1.31) (page 16). The definition of P_i depends on the definition of its two terms R_i and E_i. We will thus now define their values.

```
{ "env_score" :[
    { "natural":"water",          "score": 90.0 },
    { "natural":"wetland",        "score": 95.0 },
    { "natural":"beach",          "score": 70.0 },
    { "landuse":"basin",          "score": 80.0 },
    { "landuse":"commercial",     "score": 10.0 },
    { "landuse":"farmland",       "score": 70.0 },
    { "landuse":"farmyard",       "score": 60.0 },
    { "landuse":"forest",         "score": 80.0 },
    { "landuse":"garages",        "score": 30.0 },
    { "landuse":"grass",          "score": 70.0 },
    { "landuse":"industrial",     "score": 3.0 },
    { "landuse":"landfill",       "score": 0.0 },
    { "landuse":"meadow",         "score": 80.0 },
    { "landuse":"residential",    "score": 55.0 },
    { "historic":"castle",        "score": 80.0 },
    { "historic":"city_gate",     "score": 80.0 },
    { "historic":"citywalls",     "score": 80.0 },
    { "historic":"*",             "score": 70.0 },
    { "building":"farm_auxiliary","score": 70.0 },
    { "building":"farm",          "score": 70.0 },
    { "building":"barn",          "score": 70.0 },
    { "building":"commercial",    "score": 20.0 },
    { "building":"industrial",    "score": 10.0 },
    { "building":"cathedral",     "score": 90.0 },
    { "building":"chapel",        "score": 80.0 },
    { "building":"church",        "score": 80.0 },
    { "building":"construction",  "score": 30.0 },
    { "building":"garage",        "score": 30.0 },
    { "building":"ruins",         "score": 90.0 },
    { "building":"*",             "score": 40 },
    { "leisure":"garden",         "score": 80.0 },
    { "leisure":"golf_course",    "score": 70.0 },
    { "leisure":"marina",         "score": 70.0 },
    { "leisure":"park",           "score": 70.0 },
    { "leisure":"bird_hide",      "score": 95.0 },
    { "tourism":"picnic_site",    "score": 80.0 },
    { "tourism":"artwork",        "score": 80.0 }
  ],
  "default_env_score" : 60.0
}
```

Listing 2: Example of profile characterising the value of the environment

```
{
  "road_scores": [{"..."}],
  "env_scores": [{"..."}],
  "alpha": 0.5,
  "forbidden_roads": [{"highway": "motorway"}, {"highway": "motorway_link"},
↪  {"highway": "motorway_junction"}, {"highway": "trunk"}, {"highway":
↪  "trunk_link"},{"highway": "primary"}, {"highway": "primary_link"},
↪  {"highway": "steps"}]
}
```

Listing 3: Example of user profile.

Score for the road in itself

Value R_i is obtained by simply summing up the preference values of all tags:

$$R_i = \frac{\sum_{j=1}^{m} s_{ij}}{m} \tag{2.2}$$

where the set $\{1, ..., m\}$ represents the different tag keys (example: `highway` and `surface`) defined in the profile; s_{ij} is the score associated for the tag key j to the road segment i. Thus, m is the number of tag keys in the profile ($m = 2$ in the listing 1 since we only have `highway` and `surface`). Scores in the profile are comprised between 0 and 100 and so is R_i.

Score for the road's environment

For the value E_i associated to the environment it is a little more complicated, since the notion of environment must be defined. Indeed, the road environment can be seen as elements at a distance lower than 20 meters of the road but this can be also elements at a distance greater like a mountain, a large field, etc. In this work, we limit ourself to the immediate surroundings of the roads (less than 20m). In future works, it would be interesting to extend this definition to a wider vision of the surroundings.

To assess the quality of environment, we create a *road surrounding polygon* (rsp_i) which is defined by the area at a distance of at most 20 meters of the road segment i as shown in figure 2.4. We then intersect this rsp_i polygon with each polygon p_j from OSM data in order to compute the area of overlapping o_{ij}:

$$o_{ij} = area(rsp_i \cap p_j) \tag{2.3}$$

We also define an *unknown area* u_i for each road segment as follows :

$$\text{unknown area} = u_i = \max(0, area(rsp_i) - \sum_j o_{ij}) \tag{2.4}$$

The unknown area is intuitively the area of the part of the road surrounding polygon rsp that no polygon overlaps. In the figure 2.4, this is thus the area of the whole rsp_i minus the area that intersect the polygon p_j, that is to say o_{ij}. As you can see in the figure 2.5, the definition given is not exact. Indeed, in the case where we have two polygons that overlap each other, the unknown area u_i is the area of the whole rsp_i minus the area of the two polygons. Both of these areas contains the striped part, that is thus counted twice. The sum of the o_{ij} can

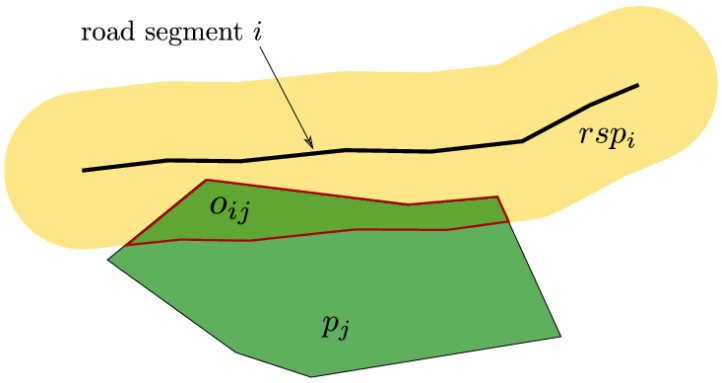


Figure 2.4: Road surrounding polygon (rsp_i, in yellow) for a road segment i (in black) and intersection with a polygon p_j (in green) to form o_{ij} (outlined in red).

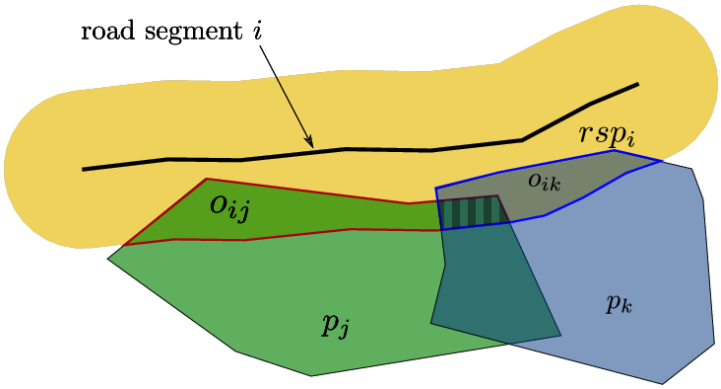


Figure 2.5: Problematic overlapping between o_{ij} and o_{ik}

thus be greater than the area of the round surrounding polygon rsp_i. We use the maximum function to force the value of the unknown area to be positive. A more precise definition of the unknown area can be computed by using an order of precision between polygons (for instance: building are more precise than residential area, river are more precise than forest,...) and by taken into account for the overlapping part only the area of the more precise polygon. After some tests, we have seen that the computation needed to do this more precise definition is too expensive (intersections and subtractions of polygons being the critical parts). Consequently, we have simplified our definition. This definition being less precise we can ask why it is still important to compute the unknown area. Imagine the case where we are in a poorly mapped area with only one small building drawn near our road segment. Without the unknown area, the only score that we can use is the score associated to the building but this is not representative of the real environment. In conclusion, we can do the hypothesis that in well mapped area, our unknown area has a value near 0 and that the unknown area is only useful in not well mapped environment.

Now that we have defined o_{ij} and the unknown area u_i we can use them to compute the value E_i corresponding to the environment of the road segment i:

$$E_i = \frac{\sum_{j=1}^{n} (o_{ij} s_j) + u_i s_{unk}}{\sum_{j=1}^{n} (o_{ij}) + u_i} \tag{2.5}$$

where n is the number of polygons, s_j is the score from the profile associated to the polygon j and s_{unk} is the default score from the profile associated to unknown environment.

We also want to take into account the presence of other large roads near the considered road. For instance, paths along a motorway are commonly not pleasant at all. To deal with this, we construct one polygon for each important road segment. The width of theses polygons are defined by the tag `width` if it is defined or by the tag `lanes` multiplied by 3.5 meters (that is approximately the mean width of a lane). Afterwards, theses polygons can be integrated in our definition proposed before as any other polygons. Since rivers and railways are also represented by lines in OSM and that these elements are also important for the computation of the pleasant value, we create corresponding polygons. In a further work it can be interesting to study the possibility of taking all lines elements directly into account.

The definition of the P_i is now complete. We can still note that it is possible to tune precisely this value with our profile. Indeed, we can give a greater (or a lower) importance to particular tag key by adapting his variance. Indeed, a tag key that has his score between 20 and 80 impact more than a tag key that has his preference values between 45 and 55. And the parameter α allows to easily define the importance between the two kind of preference.

Remark. *The attentive reader can note that the rsp_i of consecutive road segments overlap each other at the end. We have done the hypothesis that this does not affect so badly our computation but in further work, it can be interesting to study its real impact and to improve our model if needed.*

2.5 Conclusion

We have seen in this chapter that the pleasure associated to a road is very difficult to quantify. It depends from a user to another, of the mode of transportation but also of multiple external factors like the weather. Furthermore, the available data on Internet are limited due to the difficulty of representing the complex reality of the world and the difficulty to harvest all (and to keep all up to date). We are also limited by the complexity of the computation. For instance, this complexity forcing us to simplify the unknown area definition. Moreover, some part of the computation can be done in preprocessing but given the size of the data this preprocessing must remain sufficiently efficient. As we want to take into account the preferences of the user through a profile, some computation can only be done once the user has sent his request and it is thus important that theses computations are done very quickly. In conclusion, this work cannot do a perfect assessment of the pleasant value of roads. We expect however that this approximation will be sufficient to propose quicly a nice walk to the final user based on his own preferences.

We have also seen in the state of the art that our approach is not the only one. In further work, it can be interesting to integrate some of theses approaches in our computation. For instance, it is possible to extend our model of E_i to take into account the density of geotagged photography. Another things that can be improved in further work is the creation of the profile. Indeed, we request to the user to provide a user profile as complete as possible. Using techniques of machine learning to create the profile of a user based on walks that he has done before can thus be very interesting.

Now that we have analyzed and defined the two main problem behind our application, namely the routing optimization problem and the assessment of a pleasant value for a road segment, the next chapter is dedicated to the effective implementation of our application.

Chapter 3

Architecture, algorithms and implementation choices

The goal of this master thesis is finally to propose a web application usable by everyone. For instance, if you want do a walk with some friends you can use the application. You indicate your location, the desired distance of the tour and some preferences like the mode of transportation and the kind of terrain that you like. Then you click on the search button and a pleasant tour is computed for you.

The purpose is thus to propose to the user an application that finds a tour as pleasant as possible, according to its own preferences. Because this final application depends on different modules that interacts with each other, the first section presents the architecture and the different modules. After that, sections 3.2 to 3.6 describe more in details these different modules, their implementation choices and the algorithms used.

3.1 Architecture overview

Figure 3.1 shows an overview of the architecture of our application and the different modules that are involved. The left part of the figure is dedicated to the preprocessing phase whereas middle and right parts are about processes that run when a user make a search.

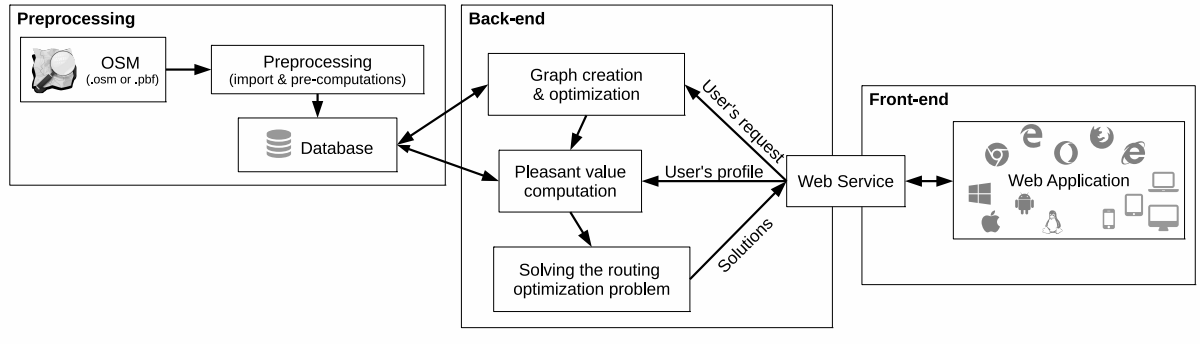


Figure 3.1: Architecture overview

The application is therefore separated in three main components represented by the three main boxes in the figure:

(i) **Preprocessing:** This component is responsible for importing data from OpenStreetMap (OSM) and does computations that are independent of the user's request and preferences. It is also responsible to keep the imported data up to date. The figure shows that these data are imported from **OSM**, then we do some conversions and computations on it (**Preprocessing** phase) and finally we store results in the **Database**. Section 3.2 explains the preprocessing phase, the difficulties encountered with the data, the database choices and which are computations that can be already done at this stage.

(ii) **Back-end:** This component is responsible for solving the main problem of the application. That is to say the routing optimization problem defined in the chapter 1 based on pleasant values defined in the chapter 2. This part of the application can be decomposed in smaller parts represented by three boxes in the figure:

   (a) **Graph creation and optimization:** The first step of the back-end is to obtain a road graph. Based on the user's request, we retrieve from the database the data related to the useful road segments in order to create the graph. Then we optimize it by removing unnecessary arcs to reduce the search space. The section 3.3 explains how we have implemented this part of the back-end.

   (b) **Pleasant value computation:** The second step in the back-end, described in details at the section 3.4, is dedicated to the computation of the pleasant value for each road segment as defined in the section 2.4.3 (page 29).

   (c) **Solving the routing optimization problem:** The last step of the back-end is finally to solve the problem presented in the chapter 1 to produce solutions to our problem. These solutions are based on the graph and the pleasant values computed at the previous phases. Section 3.5 explains how we tackle the problem, how we get a first solution and how we improve it.

(iii) **Front-end:** This last component contains the user friendly **web application** that anybody can use easily. This web application communicates with the back-end through a **web service**. Section 3.6 describes the front-end interface and how we have constructed the communication between this interface and the back-end part of the application.

We decided to implement the most these components in the Scala programming language except the database and the web application part that are obviously in other dedicated languages.

3.2 Preprocessing phase

The purpose of the final application is to find a pleasant tour sufficiently rapidly. The user that search would not like wait for a couple of hours to have a tour. To deal as better as possible with this additional constraint, computations that are possible without knowing user's request and preferences are done in a *preprocessing phase*. This phase is done once for all users and has for purpose to prepare data and compute interesting values on these data. The result of the computation phase is finally stored using a database as described below. In this way, the preprocessing phase doesn't take time when the user starts a search.

To solve our problematic rapidly, we have to construct quickly a graph representing all road segments that are at a given distance of a given location. Associated to this graph, a pleasant value for each arc must be defined. Since the computation of the pleasant value takes some times, and particularly the computation of the area of intersections between polygons, we would like compute it for of all roads segments one time and store it to quickly access to them after. Nevertheless, the final pleasant value of earch road segment cannot be computed a priori because user's preferences must be known. However, it is possible to compute intermediate values, that is to say, the area o_{ij} of intersection of each polygons near each road segment i. For this data must also be prepared. Indeed, we have to segment ways representing roads in smaller *road segments* that link two crossroads. We want also to convert the representation of the polygons in the data to a more easily usable representation.

3.2.1 Database choices

As we have to store data and values that we retrieve and compute from the OSM data, we must decide the way of storing these data. Databases seem to be the better choice because they allow us to manage and access to the data by simply doing requests. The choice of the database system is however less obvious.

After making some researches and tried some database systems we decided to use a *PostgreSQL* database system. This choice has been done for the following reasons. First, this is the database system used by OSM itself, and thus this indicates that PostgreSQL is sufficiently efficient to treat with the large amount of data contained in OSM. Secondly, the extension *PostGis*[1] adds to PostgreSQL useful functionalities to treat with geometry and geographical elements like points, polygons and lines. Moreover, PostGis adds also the spatial indexing functionality that allows us to retrieve rapidly data based on their geographical location. The use of this extension also allows us to use the QGis[2] software to visualize the content of the database. Finally, the last version of PostgreSQL (PostgreSQL 9.5) adds the `UPSERT` feature that is a special insert doing update if the row key exists already[3]. This functionality seems interesting to keep data up to date.

Since PostgreSQL allows us to construct functions and trigger, we put all necessary computations done in the preprocessing phase inside functions in the database. In this way, computation functions can access to the data very quickly. The import of OSM data consists thus simply to download and put data inside the corresponding tables in the database and to call the computation functions that are inside the database.

Database schema

Our database in composed of the following tables that can be directly populated from the raw OSM data:

- `node`: Contains the location of OSM nodes, their IDs and their versions (useful for keep data up to date).

[1] `http://postgis.net/`
[2] `http://qgis.org/`
[3] See `https://wiki.PostgreSQL.org/wiki/UPSERT` for more details.

- `way`: Contains the IDs of existing OSM ways and relations[4], their version and two boolean indicating if it is a road and if it is a polygon. To make the distinction between ways and relations, we add the negative sign for the ID of relations.
- `way_tag`: Contains the tags associated to ways and relations.
- `way_node`: Contains the nodes IDs that are involved in ways and their positions in the way.
- `relation`: Contains for each relation, the way IDs that are involved in the relation and a boolean indicating if the way is considered as an inner way or as an outer way.

From these tables, we can compute additional information to populate the following tables. The purpose of these tables is clearly to make our final application more efficient when the user makes a request.

- `road`: Contains road segments information. A road segment is the part of a way that crosses no other way (except at its ends). This table contains the way ID, the positions in the way from which the road segment begins and ends, the distance of the road segment and the road segment buffer, namely the locus that is at a distance less than 20m of the road.
- `polygon`: Contains the geometry of the polygons and their IDs.
- `road_env_area`: Contains areas of intersection between a road segment buffer and surrounding polygons (see equation (2.3), page 31).
- `road_env_area_unk`: Contains the unknown areas for each road segment (see equation (2.4), page 31).

Database function

The four last tables presented in the previous sections are populated by functions in *SQL* and in *plpgsql* created inside our database. These functions are the following:

- `create_roads`: The purpose of this function is to create missing road segments and to ensure that road segments intersect each other only at their extremities. Listing 4 shows how we have implemented this function. First, we add all ways that are roads and that are not yet in the `road` table. Then we cut road segments that intersect other road segments in their inside nodes. Finally, we recompute the distance and the road segment buffer of updated road segments.
- `create_polygons`: This function is responsible to populate the `polygon` table by creating the geometry from ways and relations that are polygons. This function is divided in three parts. First, we compute non-existing polygons associated to relations. Second, we compute non-existing polygons associated to ways that are polygons and finally, we compute non-existing polygons corresponding to important roads, rivers and railways.
- `create_roads_env`: The purpose of this function is to populate the two tables `road_env_area` and `road_env_area_unk`. This is the most critical function since they must compute intersection of polygons, which is time consuming.

[4]We import only relations that are *multipolygons*, the others are unused in our application.

```
CREATE OR REPLACE FUNCTION create_roads() RETURNS boolean AS $$
  --Step 1: Insert all ways with is_road and that aren't yet into road table.
  INSERT INTO road (osm_way_id, start_pos, end_pos)
  SELECT w.osm_way_id, min(wn_min.pos) as start_pos, max(wn_max.pos) as end_pos
  FROM way w
  JOIN way_node wn_min ON w.osm_way_id = wn_min.osm_way_id
  JOIN way_node wn_max ON w.osm_way_id = wn_max.osm_way_id
  WHERE  w.is_road
  AND NOT EXISTS (SELECT 1 FROM road r WHERE r.osm_way_id = w.osm_way_id)
  AND wn_min.pos != wn_max.pos
  GROUP BY w.osm_way_id;

 -- Step 2: Cut roads segments that intersects others at a node between extremities.
 SELECT cut_road(wn2.osm_way_id, wn2.pos) -- cut the road at the given position.
 FROM way_node wn
 JOIN road r ON wn.osm_way_id = r.osm_way_id
 JOIN way_node wn2 ON wn.osm_node_id = wn2.osm_node_id
 JOIN road r2 ON wn2.osm_way_id = r2.osm_way_id
 WHERE (wn.osm_way_id != wn2.osm_way_id OR wn.pos != wn2.pos)
 AND wn2.pos > r2.start_pos -- We cut only if the intersection is
 AND wn2.pos < r2.end_pos   -- inside a segment.
 GROUP BY (wn2.osm_way_id, wn2.pos);

 -- Step 3 : We compute missing roads line and rsp_i (called line_buff).
 WITH line_to_update as
     (SELECT
         r.osm_way_id,
         r.start_pos,
         r.end_pos,
         get_way_line(r.osm_way_id, r.start_pos, r.end_pos)::geography as line
     FROM road r
     WHERE line_buff IS NULL OR line IS NULL)
 UPDATE road r
 SET line = line_to_update.line,
   line_buff = ST_BUFFER(line_to_update.line,get_road_env_meters())::geometry,
   distance = ST_Length(line_to_update.line)
 FROM line_to_update
 WHERE line_to_update.osm_way_id = r.osm_way_id
   AND line_to_update.start_pos = r.start_pos
   AND line_to_update.end_pos = r.end_pos;
 SELECT TRUE;
 $$ LANGUAGE SQL;
```

Listing 4: `create_roads` function putted inside the database.

3.2.2 Data import

There exist several manners to retrieve data from OSM. Indeed, OSM provides an "Export" function on its website but there are also some files already prepared and some webservice API that allows everyone to download OSM data[5]. Files downloaded are in two predominant formats: the *OSM XML* file format[6] and the *PBF* file format[7]. Both can be read easily using

[5] For more information about how to download data, we invite the reader to visit `http://wiki.openstreetmap.org/wiki/Downloading_data`

[6] `http://wiki.openstreetmap.org/wiki/OSM_XML`

[7] `http://wiki.openstreetmap.org/wiki/PBF_Format`

the *OsmParser* from the scala library `io.plasmap.parser.OsmParser`[8]. Our import consists thus to use this library that reads an OSM file and to add all nodes, ways and relations (and the associated tags) inside our database by simply using `INSERT` instructions of our database. After having importing data, we call the three functions mentioned just above to populate the remaining tables. In this import process, we also determine if a way represent an area, that is to say a polygon; in order to do that, we use the description proposed in the webpage *Overpass turbo/Polygon Features* by Raifer [30].

3.3 Back-end: Graph creation and optimization

Before we can start to solve the routing optimization problem we have to prepare the road network graph based on the user's request and preferences. So, we retrieve all road segments that are within the circle centred on the user starting location and that have a radius of the maximal distance divided by 2. These road segments can easily be retrieved from the database thanks to geospatial index functionality of PostGis mentioned above. We discard from this research road segments that are forbidden by the user (e.g. steps for cyclists). Then, based on the road segments retrieved from the database we create a directed graph by adding two arcs, one in each direction, for each road segments.

When this graph is created, it is still possible to remove some useless arcs. This optimization phase allows us to reduce the search space and thus can be very beneficial. Indeed, our graph contains some dead-end streets that we want to eliminate and also some arcs that are not reachable from the starting vertex given the maximal distance. This last category of useless roads is due to the fact that we retrieved from our database road segments based on the distance as the crow flies but roads are never in a straight line.

Algorithm 1 presents how we remove the arcs. First we compute the shortest distance between v_0 and each vertex of the graph. For this computation we use the famous Dijkstra Shortest path algorithm. Then we loop over all arcs in the graph and we remove arcs for which the shortest distance is greater than the half of the maximal distance of the tour. Indeed, as we want come back to the starting point and because our directed graph contains arcs in both directions for each road segment, we can assume that if a vertex is in a greater distance it can not be reached. Finally, we take dead-end and we remove it. Because the removal of a dead-end road segment can cause others dead-end, we repeat the operation until no more dead-end are found.

3.4 Back-end: Pleasant value computation

Once the road network graph is reduced, we associate a pleasant value to each of its road segment. As discussed previously, this pleasant value depends on the profile of the user. We thus implemented the profile as an object that permits the computation of the pleasant score associated to two kinds of elements: (i) The score associated to a road segment by giving to the profile the tags of the road segment; (ii) The score associated to a polygon in the environment by also giving to the profile the tags associated with the polygon. By using the data in the

[8] `https://github.com/plasmap/geow`

Algorithm 1: Graph optimization algorithm

```
1  Function optimize(G,v_0, d_max)
2      D ← ShortestPath(G, v_0)
3      for each arc in G do
4          v_1 ← arc.getStart()
5          if D[v_1] > d_max/2 then
6              G.remove(arc)
7      deadEndArc ← getDeadEndArc(G,v_0)
8      while ∃ deadEndArc do
9          G.remove(deadEndArc)
10         deadEndArc ← getDeadEndArc(G,v_0)
11 Function getDeadEndArc(G,v_0)
12     Return the first arc in the graph that do not have v_0 as extremity and have no
       successors or no predecessors.
```

precomputed table `road_env_area` and `road_env_area_unk` and the function implemented in the profile, we are thus able to compute the pleasant value for each road segment.

3.5 Back-end: Solving the routing optimization problem

Now that the road network graph is reduced and that pleasant values are associated to its road segments, we are interested in solving the routing optimization problem that we defined in the section 1.4. A first naive approach to solve this problem can be to try each possible combinations of value for the decision variables, verifying for each combination if it is valid regarding to the constraints and finally compute the pleasant value of the tour and taking the best tour that exists. Nevertheless, this naive approach does not work in reality given the exponential number of possible paths. We need a more efficient approach in order to get solutions in a reasonable time. Given that we have a problem subject to several constraints, an approach that seems good and easy to setting up is the *Constraint Programming* (CP). The idea behind CP is to separate the *model* that describe a particular problem and the *search* for solutions solving the problem. In this way, multiple search techniques can be used for a same model and searches can be applied to multiples models. The model describe thus a real world problem, the way we want to represent it, the values we want to optimize and what are the constraints on the values of the variables. A model is thus generally composed of a set of variables and their respective domains, a set of constraints over these variables and possibly an objective function to optimize. The search is then responsible to solve the problem by browsing the entire search space. To find solutions, the search constructs a tree search where each node correspond to a decision. The search propagates constraints and use branch and bound algorithms to remove unnecessary parts of the tree search. It also uses backtracking techniques when necessary to come back on a previous decision. In this way, constraint programming allows to make a complete search in the search space. The reader that would like know more about CP is invited to consult the *Handbook of Constraint Programming* [33].

3.5.1 Constraint programming in OscaR

We choose to use the *OscaR framework* [24] to turn our problem in a constraint programming based model and to search for solutions. The two next subsections describe respectively how we define the model and then how we search for solutions.

CP model

The CP model that we use is the following. Each arc is associated to an index. For each arc, we associate a decision variable - called the *successor* - that has as possible values the index of the arc and the indices of the arcs that can follow the arc in the graph. An arc that has its own index as successor is an arc that is not taken in the solution whereas an arc that has another index is part of the solution, and the arc that follows them in the tour is precisely the arc with the index given by the successor. The listing 5 shows how we initialize successors decision variables. A decision variable *isTaken* directly linked to the successor is also initialized in order to define more easily constraints. The *isTaken(i)* variable is set to true (i.e. 1) if the arc i is taken in the tour and set to false (i.e. 0) if the arc is not taken.

```
// Successors decision variable definition
val successors = new Array[CPIntVar](n)
// isTaken decision variable definition is linked to the value of successors
val isTaken : Array[CPBoolVar] = new Array[CPBoolVar](n)

var i=n
while(i>0){
  i-=1
  successors(i) = CPIntVar(
    Set(i) ++ //The arc can have itself as successor (i.e. it is not taken)
      arcs(i).getSuccessors.map(arcsIndices.get(_).get)
  )
  isTaken(i) = successors(i) !== i //Arc is taken if its successor ≠ itself.
}
```

Listing 5: Initialization of *successors* and *isTaken* decision variables.

With this representation, it seems pretty simple to tackle the flow conservation constraint and the subtour elimination constraint since OscaR has a constraint called `SubCircuit` that must do that.

SubCircuit constraint The subcircuit constraint first imposes that all successors are different. Then, the constraint keeps up to date information about the beginning and the end of each (partial) path and the number of (partial) paths. Each time a successor is bounded to a value, if the successor is taken (i.e. different to its index), the constraint checks the number of (partial) paths. If there are more than one path, constraint removes from the end of the current path the index of the beginning of the path. Indeed, it is forbidden to close directly the current path if it exists another path [27]. Currently, the constraint *SubCircuit* is affected by an issue and it is still possible to have multiple circuit in some particular situations. This is why we have implemented an additional constraint - called `UniqueCircuit` - that raises a Failure when

multiple subcircuits exist. Since `UniqueCircuit` constraints only checks when all successors are bounded, it can be interesting in further works to fix the *SubCircuit* constraint to do better pruning.

With the *isTaken* variable, the total length of a solution can be easily computed as

```
val totalLength = weightedSum(lengths,isTaken.asInstanceOf[Array[CPIntVar]])
```

And constraints over this distance are simply expressed as

```
add(totalLength >= minDist)
add(totalLength <= maxDist)
```

To impose that the tour quits the starting vertex, we add the following constraint:

```
val v0OutgoingArcs = v0.getOutgoingArcs.map(oa =>
  arcsIndices.getOrElse(oa,throw new Exception("invalid outgoing arc"))
)
val v0OutgoingArcsTaken = v0OutgoingArcs.map(isTaken(_)).toArray
```

Finally, we are interested in computing the objective function as described in the definition of the problem. To do this, we group arcs by their associated road segment (recall that a road segment is represented by two opposite arcs). We then assign a decision variable to each road segment, that is true if the road segment is taken in its two directions and false otherwise. These decision variables are based on the `isTaken` decision variables and initialized as follows:

```
val roadSegments = arcs.groupBy(_.getRoadSegment).toArray
val roadSegmentsIsTakenTwice = roadSegments.map{case (rs,a) => {
  assert(a.length <= 2, "Invalid graph: more than 2 arcs for a road segment.")
  a.length match {
    case 2 =>
      val idx1=arcsIndices.getOrElse(a(0),throw new Exception("invalid arc"))
      val idx2=arcsIndices.getOrElse(a(1),throw new Exception("invalid arc"))
      isTaken(idx1) && isTaken(idx2)
    case _ =>
      CPBoolVar(b = false)
  }
}}
```

With these last decision variables we can now compute our objective function, namely the pleasant value of the whole tour. This total pleasant value is computed by adding the following additional piece of code :

```
// Opposite of pleasant values if positive and 0 otherwise.
val roadSegmentsOppositePleasantValues = roadSegments.map{case (rs,_) =>
  -Math.max(0, rs.getPleasantValue(profile)).toInt
)
val totalPleasantValue = weightedSum(
  pleasantValues ++ roadSegmentsOppositePleasantValues,
  (isTaken ++ roadSegmentsIsTakenTwice).asInstanceOf[Array[CPIntVar]]
)
```

Maximizing the pleasant value is expressed as follows:

```
maximize(totalPleasantValue)
```

CP search

Now, that the routing optimization problem is described inside a constraint programming model, it is possible to use it in order to search solutions. Multiple generics branchings strategies are already implemented in the OscaR framework. Nevertheless, it is also possible to implement a new search strategy in order to take profit of the properties of our problem. For this reason we decide to implement a new branching - called *SubCircuit Branching* - with some variants (*with come back* and *with objective*) that make the search for a first solution faster.

SubCircuit Branching: The principle of this (binary) branching is pretty simple. While no arc leaving the starting vertex $v0$ is already taken[9], we take randomly one of the successors corresponding to an arc that leaves the starting vertex and we branch on it randomly[10]. Then, when an arc that leaves v_0 is taken, we follow the current path by iterating over the bounded successors (starting from the starting arc) and we branch randomly on the first unbounded successor.

SubCircuit Branching with come back (SubCircuitCB): This variant works as SubCircuit Branching except in the choice of the value to assign to the unbounded successor. Indeed, when the current path has a length greater than the half of the maximum distance for the tour, this branching selects the value of the successor that minimizes the distance to the v_0.

SubCircuit Branching with objective (SubCircuitObj): This variant works as the SubCircuitCB but while the length of the current path is lower than the half of the maximum distance, the branching selects the value for the successor that minimizes the distance to the arc that has the best pleasant value and for which the successor is not yet bounded.

With theses branching strategies we can obtain a first solution. Since the first two search strategies use random assignation, it is sometimes useful to restart the search if the number of fails is too large. Section 4.3 presents and discusses some experiments about the different search strategies.

Once we have a first solution, we want to improve it. For this we use a technique called *Local Neighbourhood Search* (LNS). As explained by Shaw [35], the principle of LNS is the following. Based on the best solution we currently have, we relax some variables and we re-search for solutions in this limited search space. OscaR proposes for that a `startSubjectTo` functionality that permits adding some constraints for a particular search. The simplest relaxation strategy is just to randomly keep a given percentage of the solution. But it is also possible to decide to relax all successors that are taken and a given percentage of the successors that are not taken. Or also to relax all successors that are not taken and a given percentage of the successors that are taken. The selection of the successors to relax can also be based on their influence in the objective. In further work it can thus be interesting to use techniques like the "Cost Impact Guided LNS." [17] to improve the LNS.

Listing 6 shows how we relax all successors that are not part of the tour solution and a

[9] We consider that an arc is taken when its successor is bounded to a value other than its index.

[10] The branching consists to select a random value from the domain of the variable, the left decision is to assign this value to the variable and the right decision is to remove the value from the domain of the variable.

given percentage of the successors that are part of the tour[11]. To adjust the percentage of the successor to keep, the completeness of the search is useful. If the search is complete, it is better to relax more successors and if the search is not complete, it is better to keep more successors.

```
startSubjectTo(failureLimit=100){
  var i = model.n
    while(i>0){
    i-=1
    //We keep only a part of successors taken.
    if(bestSolution.get.successors(i) != i
      && Random.nextInt(100)<percentageToKeep){
      add(successors(i) == bestSolution.get.successors(i))
    }
  }
}
```

Listing 6: Example of LNS search where only a part of the successor in the tour is not relaxed.

3.6 Front-end: Web application and web service

Since the final purpose of this work is to have an application accessible and usable for as many users as possible, we decided to make a web application. This choice allows everybody that has a computer, a tablet or a smartphone to plan a pleasant tour. The choice of a web application and not a mobile application is mainly based on the fact that there remain people without smartphone and that people having a smartphone or a tablet can access to web applications easily. We also decided to don't work on an application that provides live routing because these kinds of applications already exist on the most smartphones. This is why we decided to propose to users to download their tour in the GPX file format. This file format is indeed the most common format for the routing purpose and it is thus compatible with the most GPS devices. In this manner, people that don't have smartphone but that have a GPS device can also take benefits from our application.

The front-end of the application is thus a web interface. The section 3.6.1 presents challenges behind this part of the application and how we thought the interface. Finally, section 3.6.2 presents how this interface communicate with the application's back-end.

3.6.1 User's interface

Designing a user's interface is a big challenge for all applications and can be the critical point. Indeed, an interface that isn't sufficiently easy to use leads frequently to unused applications. Nevertheless, an application that doesn't fit the user's expectations can also lead to frustrated users. Unfortunately, we have seen in the chapter 2 that the pleasant value is dependent from a very huge number of user preferences. If we propose a huge form with a lot of questions to the user, it is likely that the user doesn't take the time to complete it and quit our application. But,

[11] Because we use a random function to decide if we keep an item or not, the percentage is not always respected (but it is respected on average).

if we propose only a limited number of options, users that would like refine their preferences and thus that would take advantage of the application for their specific usage will be frustrated. We have finally opted for an interface that allows everyone to search quickly by using predetermined preferences to compose their profile but that also allows users that want a more tuned application to refine their preferences. These predefined preferences are of two kind. The first kind of preferences is about the road and is influenced by the mode of transportation (*foot* or *bicycle* in the current version). The second kind of preferences is about the environment (*Forests*, *Lakes & rivers*, *Fields & plains* or *Ruins & historical*). While the first kind allows to construct the `road_scores` and `forbidden_roads` part of the profile, the second kind allows to construct the `env_score` part of the profile (see listings 1 to 3).

Moreover, we have opted for a website without any authentication mechanism because it's seem boring and unnecessary to need a registration for just saving some settings. As shown by the figure 3.2, we opted for a system that proposes to the user to save they preference just by providing an identifier (like a username but not necessary) and a name for the preferences (for instance *"My forest profile"*). To reload saved preference, the user has just to provide its identifier and to choose in the preference's name to reload[12]. The advantage of this system is that we avoid having a boring registration system but we keep the possibility for the user to save his preferences. We propose also a custom URL to share the tour obtained with friends or to access it later.

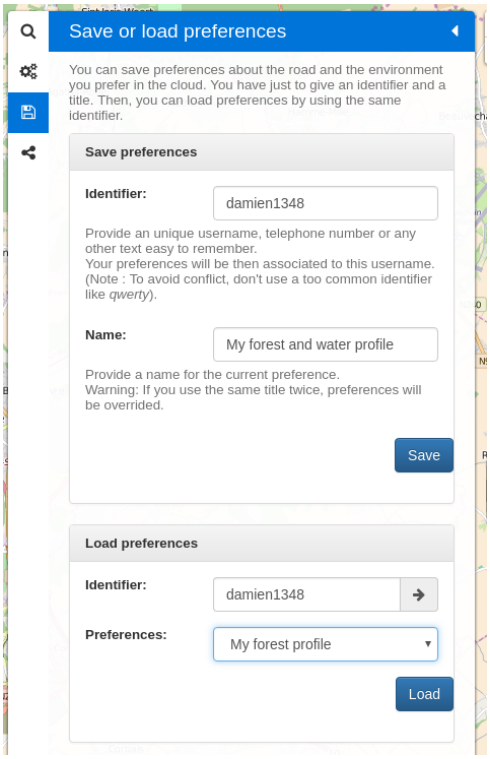


Figure 3.2: Screenshot of the *Save or load preferences* tab

Because many people are familiar with the existing routing web application like Google Maps or OpenStreetMap, we decided to design a similar layout. Our application is thus composed of a main webpage that displays a map of the world and at the left a search field that allows users to make the search request as shown in the figure 3.3. For the map, we use the javascript library

[12]This idea come from the system proposed by the search engine DuckDuckGo.com (`https://duck.co/help/settings/cloud-save`)

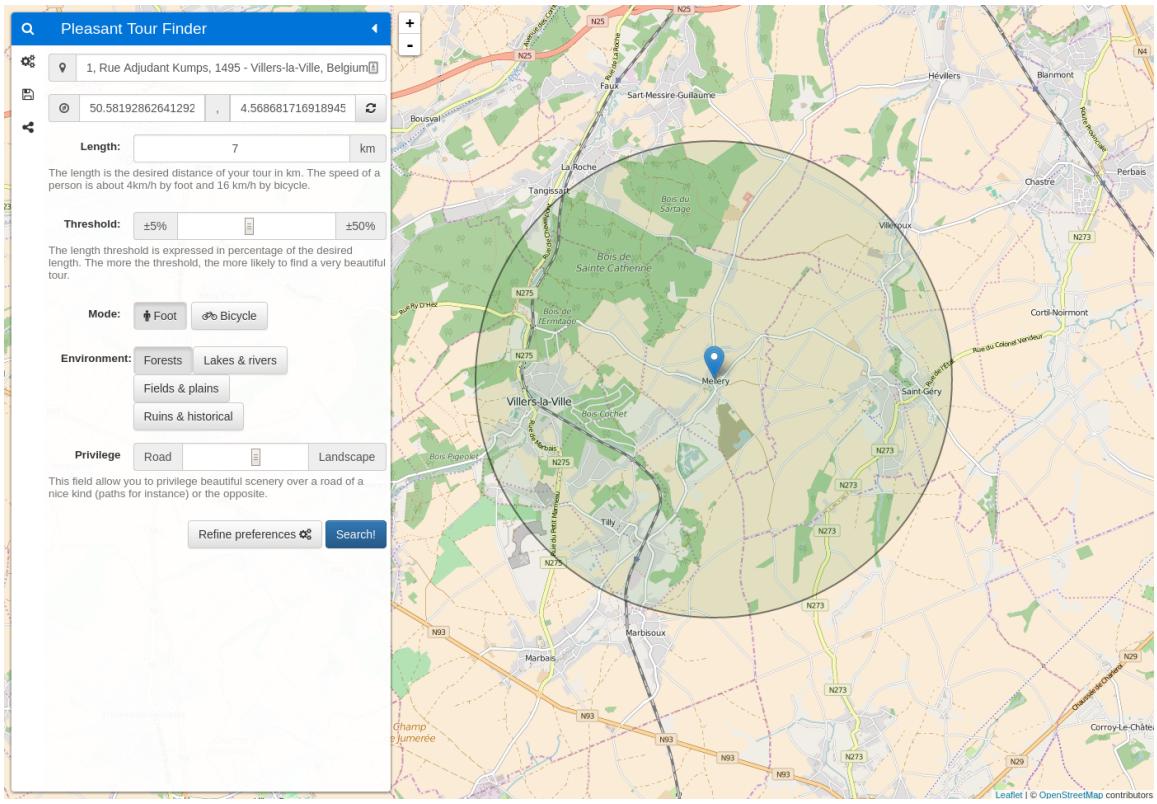


Figure 3.3: Screenshot of the main page of the application.

Leaflet[13] that has the advantage to be mobile-friendly and that proposes many useful features and plugins. We obviously use the OpenStreetMap layer for the map rendering. We add also a marker on the map that the user can move to precise the search and we mark the search space by a circle around this marker. This help the user to visualize his search request before he sends it. Then, when the user has sent his search request he can visualize the progression of the search and intermediates solutions as shown in figure 3.4. When he evaluates that the solution is good enough, he can also stop the search and enjoy his pleasant tour.

Since the front-end is mainly composed of one HTML page styled with CSS and interacting using Javascript, we decided to do not base them on a complete but complex web framework like django or Symfony. Nevertheless, we use some javascripts and CSS libraries like *leaflet* mentioned before but also *Bootstrap*[14] and *JQuery*[15].

Because this front-end must communicate with the back-end to make the search, we implemented a *web service* that is presented in the next section.

3.6.2 Webservice: Communication with the back-end

While we need to communicate with the backend part of the application that is in Scala, we have implemented a webservice (in Scala too) that responds to the HTTP requests done by the front-end. The webservice can receive multiple kind of request and some of them take some time. For this reason, we have to use multi-threads otherwise the webservice cannot handle

[13] `http://leafletjs.com`
[14] `http://getbootstrap.com`
[15] `http://jquery.com`

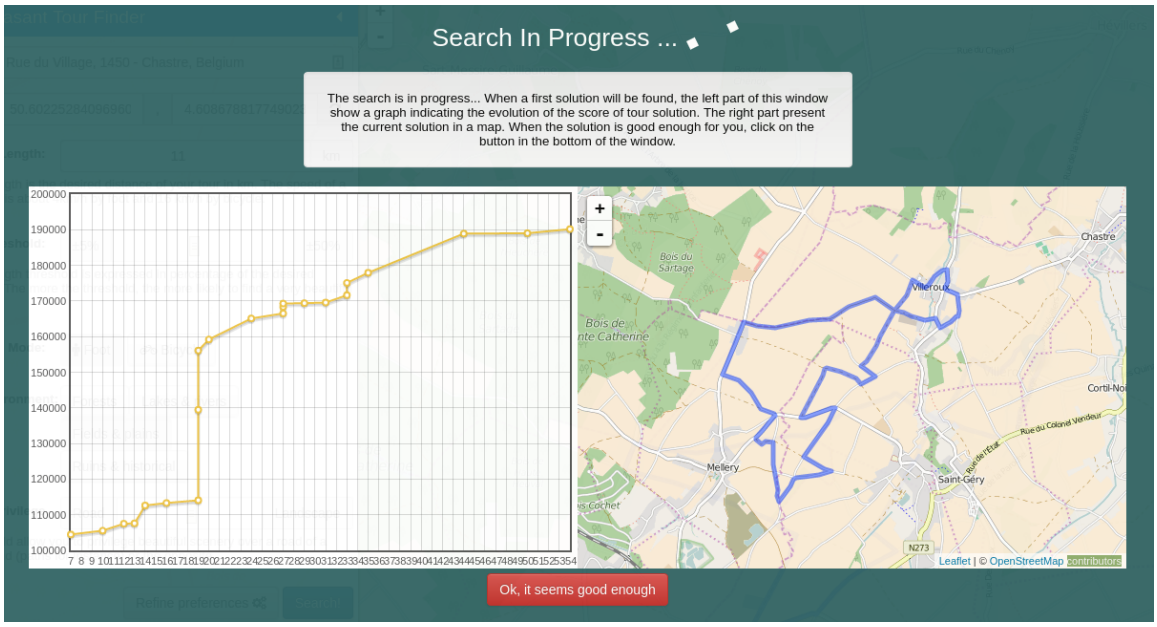


Figure 3.4: Screenshot of the page displayed during the search for a solution. The left graph shows the evolution of the solution and the right map shows the last solution (the best one).

requests of another user while it is currently responding to a first user request. Because we display intermediate solutions (see figure 3.4), we proceed as shown in the figure 3.5

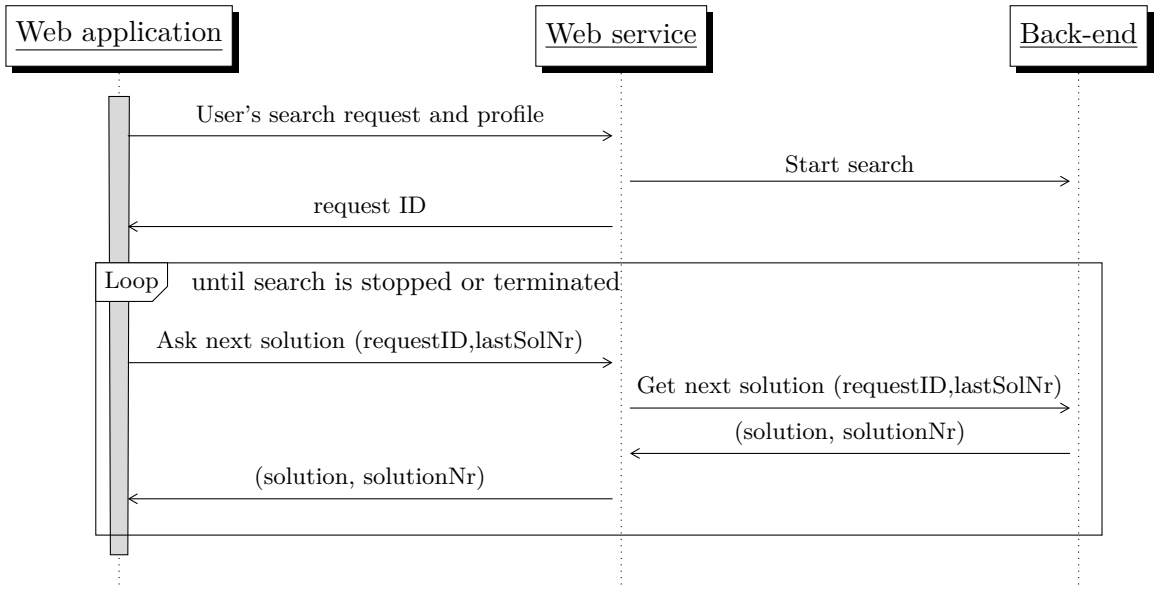


Figure 3.5: Communications between the web application, the web service and the back-end when user launches a request.

1. First, the user sends his request by sending his starting location, the desired distance and his preferences.

2. Then, the webservice handles the user's request, asks to the back-end to start the search in a separate thread and replies to the web app a unique identifier associated to his request. The usage of a separated thread for the search is used to respond quickly to the user without waiting the end of the search.

3. Then, the front-end asks for the first solution by asking the next solution for his request and indicating that he doesn't know yet a solution (lastSolNr set to 0).

4. The webservice waits while no better solution are found and then replies the last solution.

5. The solution retrieved is displayed and the next solution is asked to the webservice by indicating the number of the last solution retrieved.

6. Finally, when the search is finished the webservice indicates this to the web app and the best solution is displayed in the main map.

3.7 Conclusions and further work

We have seen in this chapter that our final application is based on multiple components that work together to propose an easy-to-use application to the final user. Given the number and the variety of components, we are focusing in a first time on the most critical parts of its. Then we assembled each components together to finally obtain our application. It remains few points that can still be improved for each components of our application.

As the most critical part for the user is currently the research for a solution, it seems important to improve in a further work the CP model and the constraints that are set on it. In particular, it is possible to develop an additional constraint based on a lower bound of the distance between each arc (for instance the distance as the crow files). An idea that could result in a good pruning is to keep in memory successors that are part of the tour (partial paths). Each time a new successors is added to the tour, it is possible to construct the set of all vertices (or arcs) that can be connected to the beginning of a partial path and to the end of another partial path in a distance lower or equal to the maximal remaining distance and to remove all vertices (or arcs) that are not part of this set. This set of vertices (or arcs) is in fact a construction of multiple ellipse with as focal points the end and the begin of partial paths. The advantage of this kind of pruning can be to quickly remove many successors from the search based on the distance. It seems thus an interesting point to develop and analyze in further work.

Another point that can be improved is the time needed to compute the pleasant value. Indeed, currently, this computation is done in the back-end by Scala code. Nevertheless, retrieving from the database area and tags necessary for the computation of the pleasant value takes some time (in particular for dense city with a lot of small polygons like in Louvain-la-Neuve). An improvement can be to send the user profile inside the database and to create a function inside the database to make this computation directly on it then retrieving the final pleasant value.

Other improvements can also be done to add functionalities for the user. For instance, we could propose to the user to interact with the search to dynamically impose some forbidden roads or some required roads. It is indeed possible to add these constraints on the fly on the CP model and continue with an adapted LNS search.

Chapter 4

Experimentations

This chapter is dedicated to some experimentations done on the components of our application. The goal of these experimentations is to validate the techniques used. Experiments can also be seen as a starting point to suggest improvements that can be done in a further work.

First, section 4.1 presents an experiment about the construction and the optimization of the road network graph. Second, section 4.2 presents experiments about the value obtained for the pleasant value based on predetermined profiles. Finally, section 4.3 discusses some experiments on solving the routing optimization problem.

4.1 Road network graph creation and optimization

To search for a tour of a given distance D, we load from our database a road network graph centered on the user's starting location and with a maximum radius of $R = \frac{D}{2}$. Then, we reduce the number of arcs in this graph by removing all unnecessary arcs as explained in section 3.3.

The first experiment shows the evolution of the number of arcs in the road graph for different radius R for 10 different locations in Brabant Wallon in Belgium. Figure 4.1 illustrates the average number of arcs before and after the optimization of the road graph (described at section 3.3). The blue line represents the average of the percentage of arcs kept in the graph, namely the number of arcs after optimization divided by the number of arcs before optimization. This percentage seems to stabilize at a value just below 90%.

Because the area increases in function of the square of the radius, the number of arcs is very likely to increase quadratically as well. Figure 4.2 shows that the arc's number per km^2 tends to be constant for an increasing radius. This number of arcs per km^2 is more important for small graphs because our starting locations are often mainly localized inside cities or villages where the density of road segments is more important.

This large amount of arcs can be an issue if the user searches for long tour. In particular, this can be a major issue for cyclists that would make probably larger tour than pedestrians. To deal with this issue it can be interesting to discard some parts of the road graph based on its characteristics. Indeed, the number of arcs impacts the time to construct the graph, the number of pleasant values to compute but also the search for a tour in this graph. In further

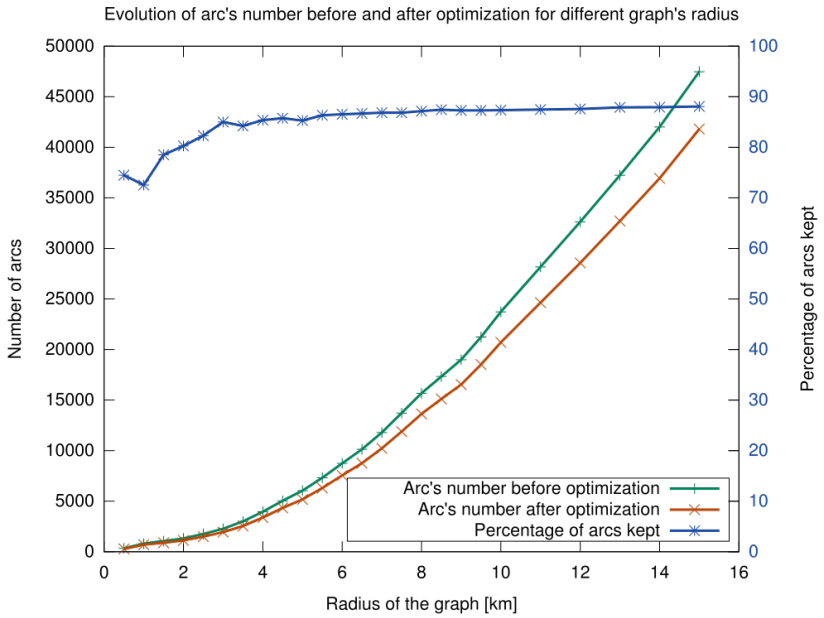


Figure 4.1: Results of graph optimization for different radius.

work, it seems thus important to develop methods to discard parts of graph even tough they can also remove some solutions. For instance, an idea to remove some parts of the graph can be to construct a grid on the map. Then a particular pleasant value based on the user preferences and on polygons can be computed for each cell. Finally, roads that are in cells with a bad pleasant value can be removed from the graph.

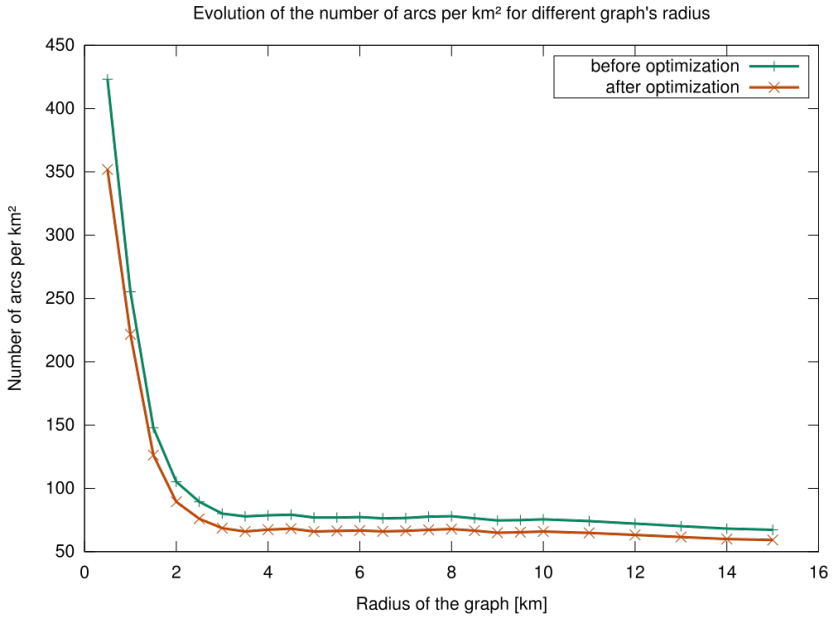


Figure 4.2: Evolution of the number of arcs by km^2 in the graph in function of the radius R.

4.2 Pleasant values assessment

For the pleasant value, it is more difficult to provide experiments because of the subjectivity of this value. Indeed, the computation of this value is based on the preferences of the user. Fortunately it remains possible to analyze predetermined profiles to see if they respond to our expectations.

As presented in section 3.4, the pleasant value is a sum of two main terms R and E - associated respectively to the road and to the environment. These two terms are weighted according to a parameter α. To easily analyze the resulting values, we create resulting maps based on the R term only (by setting up α to 0) and resulting maps based on the E term only (by setting up α to 1). Our application proposes two kinds of predefined preferences as explained in section 3.6.1[1]. The first one is preferences about the mode of transportation and can be `foot` or `bicycle`. The second one is preferences about the environment and can be *Forest* (`forest`), *Lakes and rivers* (`water`), *fields and plains* (`fields`) or *ruins and historical* (`historic`). The first kind influences the part of the user profile that is used to compute R whereas the second kind influences the user profile that is used to compute E.

To see if pleasant values correspond to what we expect, we colorize each road segment on a map. The color of each road segment then represents its pleasant value, divided by its length[2]. Road segments with a pleasant value of -50 are shown in red whereas road segments with a pleasant value of 50 are shown in green. Intermediate pleasant values follow a gradient between these two colors. The resulting map can then be visually analyzed based on our knowledge of the region and natural intuition.

Resulting maps for `foot` and `bicycle` ($\alpha = 0$)
The resulting maps for the `foot` mode and for the `bicycle` mode are shown in figures A.1 and A.2 (in appendix A, page 65). First, independently of the pleasant value, we can note that the road network graph for foot differs from the one for bicycle in some roads segments. For instance, it is allowed for a bicycle to follow all the *N4* road but some segments are explicitly forbidden for pedestrians. Other example of access restriction can be seen in the *bois des rêves* (in the center of the map) where some paths are reserved for foot and some other paths are reserved for bicycle. Second, we note that differences between pleasant values are relatively small and we do not have roads with a very high or a very low pleasant value. This low variation is because our predefined profiles contain multiple tags that affect the pleasant value. Indeed, we do not take into account only the kind of the road (`highway` tag) but also the road surface (`surface` tag) or the restriction of access on it (`foot` or `bicycle` tag). Since the `highway` tag is always present in OSM data, this is not the case for others tags, thus it is more difficult to deal with it. Thirdly, because the two modes of transportation privilege small roads over large roads, differences between the pleasant values of theses two modes are very small. Cyclists are just generally a little more tolerant on the size of the road.

In conclusion, we can see that resulting maps reflect our expectation about the computation of R values. Nevertheless, in order to have a greater variance between pleasant values it can be interesting in further work to propose an additional variable indicating the weight of each tag. For instance, based on the listing 1 (page 29), it can be interesting to indicate that scores

[1] Theses predefined preferences can be used by user as a starting point to define its own preferences

[2] We normalize the pleasant value by dividing it by the length of the road segment in order to have a value between -50 and 50 that is not impacted by the length

associated to the tag `highway` must be taken into account 3 times more than the scores associated to the tag `surface`.

Resulting maps for `forest`, `water`, `fields` and `historic` ($\alpha = 1$)
The resulting maps for the four predetermined environment preferences are shown in Figures A.3 to A.6 (appendix A, pages 66 to 69). The `forest` environment profile is well defined. Indeed, roads segments that pass in forests have clearly a better pleasant value than other. For the `water` environment profile, we can see that roads segments that pass near lakes or rivers have a better pleasant value than other but the difference is very small. Indeed, the environment of a road that intersects a lake or a river is generally only a small fraction of the environment. The pleasant value is thus less impacted by the score of the water than the score of a forest. The `fields` environment profile accords better preferences to some road segments that pass in fields. But we remark also that some other fields are not detected as well. For instance, the left of the map contains fields but road segments in this area do not have a very good pleasant value. By checking in the OpenStreetMap website[3], it seems that fields in this area are not yet mapped as such. As explained in section 2.3, the OpenStreetMap data are continuously improved. We can thus expect that this lack of information disappears in a near future. Because Louvain-la-Neuve is a very recent city that do not have a lot of historic elements, we have decided to make the map on *Vaison-la-Romain* city in France for the `historic` profile. This city contains important ruins in his center as shown in the resulting map. Nevertheless, we note that the `historic` profile can suffer from the same drawback than the `water` profile. Indeed, historical elements are not always very large things and thus do not have very much impact on the pleasant value.

Finally, figures A.3 to A.5 show that industrial estates near Louvain-la-Neuve affect very badly the pleasant value. Indeed, it is not pleasant to walk in this kind of area.

Conclusions Based on these experiments, it seems that the pleasant value can suffer from some drawbacks which are interesting to keep in mind for further work. First, when the real world elements are too small or when the distance between the road and the element is too important, the pleasant value is not very much impacted. Second, it seems that the lack of information in OSM can also impact the pleasant value. Indeed, forests are generally goodly mapped, but this is not always the case for plains and fields. However, we can expect that this lack of information disappears in future. Finally, the pleasant value does not take into account nodes from OSM. While nodes are used to designate isolated trees and other punctual world elements, it could be interesting in further work to add theses elements in the computation of pleasant value.

4.3 Solving the routing optimization problem

4.3.1 Find an initial feasible solution

We are in a first time interested in finding a first solution. Indeed, LNS requires to have a feasible initial solution to begin with. In order to analyze our branching strategies (described at section 3.5.1), we compare them with some generics branchings available in OscaR for four

[3]By using the button "query features" functionality (icon is a question mark) available in the main page.

locations[4]. We launch the search until a first solution is found or until it reaches a time out of 30 min. We repeat the operation by increasing the radius of the graph[5]. When a branching strategy reaches the time out we stop the experiment for this particular branching and we continue with branchings that do not have reached the time out. This experiment is repeated for different locations having different topographies.

Figures 4.3 and 4.4 show respectively the number of nodes and the number of fails needed to reach a first solution. For instance, for the figure 4.3a, the three generics branchings (first fail, last conflict and by pleasant value) failed to find a solution for a radius of 2 km. The experiment is not done for these branchings for larger radius. The last point of each line corresponds therefore to the number of nodes browsed by the search for the corresponding branching when it reaches the time out. Previous points are the number of nodes needed to find a first feasible solution.

With these experiments we see directly that our branching strategies (SubCircuit Branching and its variants) improve the performance by significantly reducing the number of nodes that are explored in order to find a first feasible solution. In particular the *SubCircuit with objective* seems to be the best one. Moreover, the first solution found by this branching strategy is generally of better quality than the ones found by the other branchings. Nevertheless, in some cases, this branching takes a long time (more than few minutes) that is not acceptable for the user. In our application we decide to use this branching for a limited time. If this strategy fails at finding a first solution in this limited time, we use the SubCircuit branching and we restart the search after a specified number of fails until a solution is found[6]. Indeed the SubCircuit is based on a random walk and generally provides a feasible solution quickly when it is restarted.

These experiments also demonstrate the complexity of finding an initial feasible solution in some region. Figures 4.3c and 4.4c show that all generic branchings fail to find a solution in less than 30 min for a radius of 1km around the Place Saint-Barbe in Louvain-la-Neuve. This complexity is due to the fact that Louvain-la-Neuve contains a lot of very small road segments making difficult to construct a tour. We see also that having an objective can help the search to find a tour more quickly in this kind of situations.

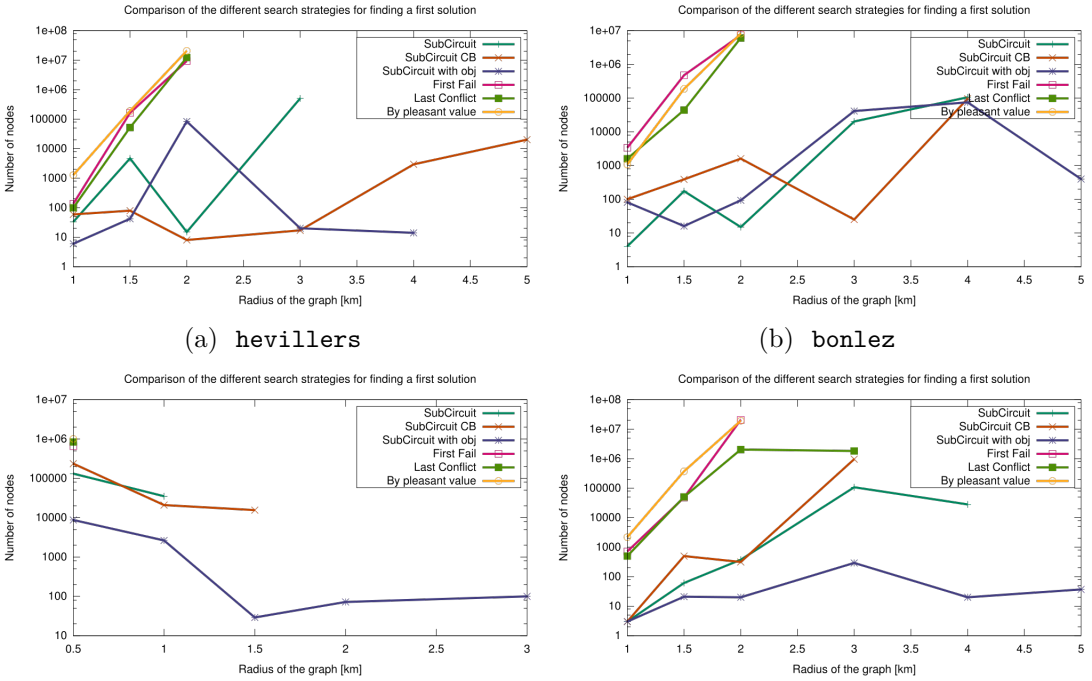
4.3.2 Quality of the solution proposed to the user

When an initial solution is found, we improve it by doing LNS as explained in section 3.5.1. But how good can we expect to be a solution obtained using LNS? We are thus interested in comparing the solution obtained by this way with the one obtained with a complete search. Table 4.1 compares the value of our objective, namely the pleasant value of the tour, obtained after using our LNS (with a maximum time of 5 min) and after a complete search. The purpose is to see if the tour proposed through the application has a pleasant value near the optimal pleasant value. Results show that values obtained by the LNS are close to values obtained by the complete search. Figure 4.5 shows the intermediate solutions obtained in function of the time. For `hevillers` and `bonlez` the time needed to reach the best value is similar for LNS and complete search. For the two more difficult locations `sainte-barbe` and `orp-jauche`, the LNS finds faster a best value than the complete search. Therefore results of this experiments

[4]These locations are named accordingly to a village or a place that is near the starting point. Exact coordinates can be found in the table 4.1.

[5]The maximum tour length is set to the two times the radius given in the graph and the minimum tour length is set to 80% of this maximum tour length.

[6]The specified number of fails is increased progressively at each restart.



(a) `hevillers` (b) `bonlez`

(c) `sainte-barbe` (d) `orp-jauche`

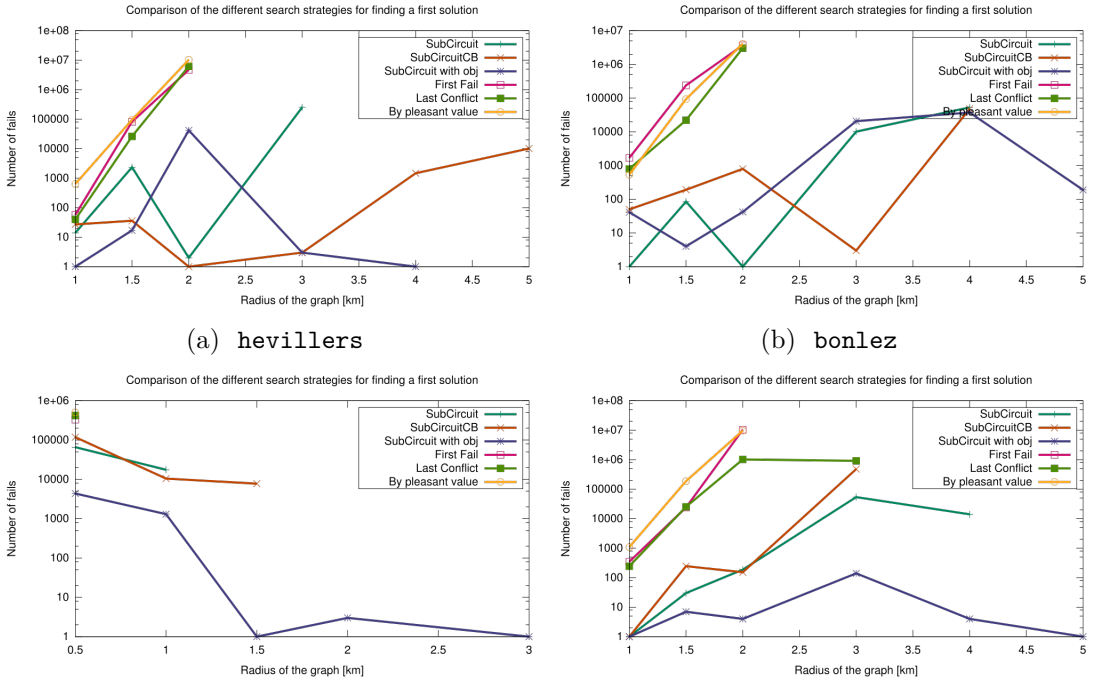
Figure 4.3: Evolution of the number of nodes for different branching strategies. Lines are stopped when the timeout of 30min is reached.

show that the usage of the LNS has a good impact on the time to find a good solution close to the optimum solution. Nevertheless, given the required time to make a complete search, this experiment is limited to small graphs without too many arcs. In a further work it can be interesting to verify if the results obtained for more larger graphs remain sufficiently good.

For the complete search we used the SubCircuit with objective branching that has shown good results for finding a first (good) solution. Nevertheless this branching requires to sort arcs by distance to an objective and takes more time to make decisions. It can thus be interesting in a further work to compare results with a complete search using another branching strategy.

	Location	Lat	Lon	Radius	Best value	Time	Gap
LNS	`hevillers`	50.62181	4.61389	2.0 km	11367.0	5.0 min	0.0%
complete					11367.0	216.8 min	
LNS	`bonlez`	50.70214	4.68966	2.0 km	35978.0	5.0 min	0.0%
complete					35978.0	5.9 min	
LNS	`sainte-barbe`	50.66826	4.62203	0.4 km	-358.0	5.0 min	0.0%
complete					-358.0	864.6 min	
LNS	`orp-jauche`	50.66426	4.94110	2.2 km	13800.0	5.0 min	0.0%
complete					13800.0	79.1 min	

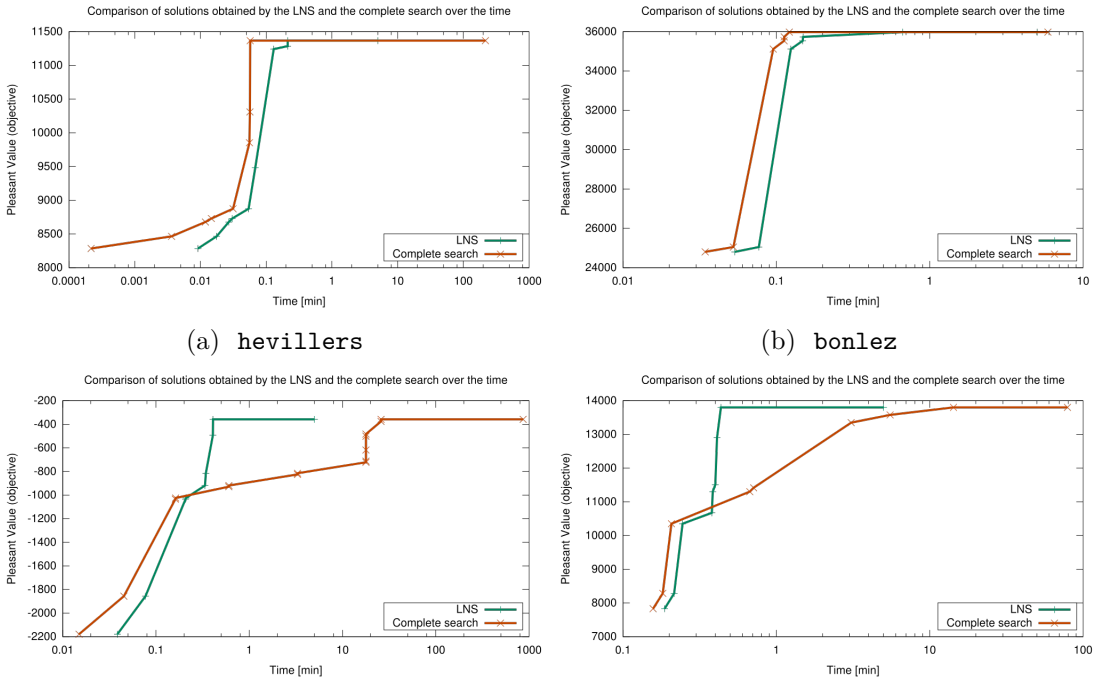
Table 4.1: Comparison of the LNS technique used in the application and the complete search (that finds the best existing solution). The last column corresponds to the gap between the best (pleasant) value obtained using the LNS v_{LNS} and the best (pleasant) value from the complete search $v_{complete}$, i.e. $gap = 1 - \left(\frac{v_{\text{LNS}}}{v_{\text{complete}}}\right)$.



(a) `hevillers` (b) `bonlez`

(c) `sainte-barbe` (d) `orp-jauche`

Figure 4.4: Evolution of the number of fails for different branching strategies. Lines are stopped when the timeout of 30min is reached.



(a) `hevillers` (b) `bonlez`

(c) `sainte-barbe` (d) `orp-jauche`

Figure 4.5: Evolution of the value of the objective (total pleasant value) for the LNS (used in the application) and for a complete search in function of the time. The time is expressed in a logarithmic scale in order to show details of the beginning of the search. The value of the last solution found is repeated when the search is stopped to indicate the total duration of the search (always 5min for LNS).

Conclusion

The purpose of this master thesis is to create an application, accessible by everyone, that proposes to the user to search for a tour that is the most pleasant as possible given constraints on starting point and on the distance and taking into account user's preferences. In order to propose such kind of application several challenges have to be tackled. To tackle these challenges efficiently, we divide our research in different parts.

In a first time, we addressed the routing optimization problem consisting in finding an optimal tour in a graph, given the pleasant values on the arcs. We first introduce some necessary basics in graph theory then we review the existing literature. Finally, because the literature doesn't provide formulation that corresponds exactly to our particular problem, we define our own formulation of the routing problem we want to solve.

In a second time, we are interested in the pleasure perceived by a person when he walks through a road segment. We focus on quantifying the pleasure felt in a *pleasant value* for each road segment, and thus each arc of the graph. To be able to define this pleasant value we first specify more precisely which is the final usage and which are the users that we target for our application. Then we study the literature to see which are the methods already used, what are their advantages and drawbacks and which characteristics can influence the pleasure of a person when he travels. We see that many characteristics can influence this pleasure and, in particular, it depends on one person to another. Because the assessment of the pleasant value requires to use data about the real world, we then analyze the available sources of data. Based on this analysis, we choose to use OpenStreetMap (OSM) as data source. Based on these cartographic data, we define in mathematical terms the pleasant value of each road segment. This pleasant value is divided in two main terms, the first one is associated to the road in itself whereas the second one is associated to the environment surrounding the road. These two terms are constructed as weighted sum of scores linked to the tags from the OSM data.

The final purpose being to create an application, we establish in a third time the architecture of the whole application. This architecture is composed of several components. Because of the application requirements we establish a preprocessing phase that has for purpose to process the data from OSM to prepare and accelerate the computation of the pleasant value and the creation of the graph. With the prepared data from the preprocessing phase, we are able to construct the road network graph and compute the pleasant value on it. Then an important and critical point is to solve the routing optimization problem based on this graph in a reasonable time. To deal with this problem we used the Constraint Programming method. This allows us to search for a first solution then to improve it using the LNS technique. Finally, the last step in the construction of our application is the user friendly interface that allows everyone to access the search. For this last step we implement two components, namely the web application and the web service. While the first one is responsible for the management of the interface to display and the interaction with the user, the second one is responsible for the communication with the

back-end.

In order to evaluate the final results of our application, we propose in a fourth time a number of different experimentations. Firstly, we see that the road graph optimization reduces from approximatively 10% the number of arcs. Nevertheless, this number of arcs seems to increase quadratically with the search radius. For long tours this can lead to difficulties to find good solutions in reasonable time. In further work it seems thus interesting to study the possibility of removing some parts of the graph, for instance by removing areas that are not sufficiently pleasant. Secondly, we visually analyze maps with road segment colorized according to their pleasant value for the different predefined preferences. Some environments seem to be more easy to take into account than others. In a further work it might thus be interesting to improve the definition of the pleasant value by adding some parameters allowing to easier accords different weights for some tags. It can also be a good thing to take into account the presence of nodes and lines in the environment. Finally, we provide experimental results on the part of our application that solve the routing optimization problem. We compare results for different branching strategies for finding a first solution. We observe that our branching strategies are better than the generic branching strategies. Nevertheless we note that the search for a first solution is not always fast enough. This is why, when our branching fails to find a solution sufficiently quickly, we use multiple restart with another branching including some random choice to find the first solution more quickly. In a further work it can be of worth to address this problem by implementing additional constraints to make more pruning during the search. Thus avoid losing time by browsing invalid solutions.

To conclude, we can claim that the final application proposed at the end of this master thesis do what we expect of it. All components interact together to provide to the user a pleasant tour through a user friendly interface. Nevertheless, each component can be improved in order to provide the solution to the user faster or to improve the quality of the pleasant value but also to provide to the user a more complete user interface allowing the user to refine their preferences in an easier way. We can also imagine extending the functionalities of our application. For instance we can add the possibility to plan multiple days tour that stop each night at a hostel or a camping. We can also extend the system to search for the most pleasant tour starting from a set of possible starting locations (for instance all railway stations in a region). Some parts of the work can also be used for other purposes. We can use the pleasant value computed to propose a route between two distinct locations that is most pleasant than the shortest path. We can also use the model defined for finding a tour that optimizes something else than the pleasant value, thereby making our application (and our model) even more general. The application proposed seems to respond to the global expectations, nevertheless it is still possible to improve each component to obtain better results, in particular in the time needed to find a tour. Components of the application can also be used as starting point to other derived applications.

Bibliography

[1] M. Alivand, H. Hochmair, and S. Srinivasan. “Analyzing how travelers choose scenic routes using route choice models”. In: *Computers, Environment and Urban Systems* 50 (2015), pp. 41–52.

[2] D. L. Applegate et al. *The Traveling Salesman Problem: A Computational Study*. Princeton university press, 2007.

[3] J. Aráoz, E. Fernández, and C. Franquesa. “The clustered prize-collecting arc routing problem”. In: *Transportation Science* 43.3 (2009), pp. 287–300.

[4] J. Aráoz, E. Fernández, and O. Meza. “Solving the prize-collecting rural postman problem”. In: *European Journal of Operational Research* 196.3 (2009), pp. 886–896.

[5] J. Aráoz, E. Fernández, and C. Zoltan. “Privatized rural postman problems”. In: *Computers & operations research* 33.12 (2006), pp. 3432–3449.

[6] C. Archetti and M. G. Speranza. “Arc routing problems with profits”. In: *Arc Routing: Problems, Methods, and Applications, MOS-SIAM Series on Optimization* (2013), pp. 257–284.

[7] M. Dell’Amico, F. Maffioli, and P. Värbrand. “On Prize-collecting Tours and the Asymmetric Travelling Salesman Problem”. In: *International Transactions in Operational Research* 2.3 (1995), pp. 297–308.

[8] M. Duckham and L. Kulik. “"Simplest" Paths: Automated Route Selection for Navigation”. In: *Spatial information theory. Foundations of geographic information science*. Springer, 2003, pp. 169–185.

[9] D. Feillet, P. Dejax, and M. Gendreau. “Traveling salesman problems with profits”. In: *Transportation science* 39.2 (2005), pp. 188–205.

[10] *Flickr.com*. URL: `https://www.flickr.com/`.

[11] M. F. Goodchild. “Citizens as sensors: the world of volunteered geography”. In: *GeoJournal* 69.4 (2007), pp. 211–221.

[12] H. H. Hochmair. *Spatial association of geotagged photos with scenic locations*. 2010.

[13] S. Kataoka and S. Morito. “An algorithm for single constraint maximum collection problem.” In: *J. OPER. RES. SOC. JAPAN.* 31.4 (1988), pp. 515–530.

[14] H. Kori et al. “Automatic generation of multimedia tour guide from local blogs”. In: *Advances in Multimedia Modeling*. Springer, 2007, pp. 690–699.

[15] T. Kurashima, T. Tezuka, and K. Tanaka. “Mining and visualizing local experiences from blog entries”. In: *Database and Expert Systems Applications*. Springer. 2006, pp. 213–222.

[16] G. Laporte and S. Martello. “The selective travelling salesman problem”. In: *Discrete applied mathematics* 26.2 (1990), pp. 193–207.

[17] M. Lombardi and P. Schaus. "Cost Impact Guided LNS." In: *CPAIOR*. Springer. 2014, pp. 293–300.

[18] X. Lu et al. "Photo2trip: generating travel routes from geo-tagged photos for trip planning". In: *Proceedings of the international conference on Multimedia*. ACM. 2010, pp. 143–152.

[19] Y. Lu and C. Shahabi. "An Arc Orienteering Algorithm to Find the Most Scenic Path on a Large-scale Road Network". In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '15. Bellevue, Washington: ACM, 2015, 46:1–46:10. ISBN: 978-1-4503-3967-4. DOI: 10.1145/2820783.2820835.

[20] J. Maervoet et al. "Tour suggestion for outdoor activities". In: *Web and Wireless Geographical Information Systems*. Springer, 2013, pp. 54–63.

[21] R. Mansini, M. Pelizzari, and R. Wolfer. *A granular variable neighbourhood search heuristic for the tour orienteering problem with time windows*. Tech. rep. Technical Report RT 2006-02-52, University of Brescia, Italy, 2006.

[22] G. Navratil. "Curviness As A Parameter For Route Determination". In: *GI_Forum* (2012), pp. 355–364.

[23] *OpenRunner.com*. URL: http://www.openrunner.com/.

[24] OscaR Team. *OscaR: Scala in OR*. Available from https://bitbucket.org/oscarlib/oscar. 2012.

[25] *Panoramio.com*. URL: http://www.panoramio.com/.

[26] G. Pataki. "Teaching integer programming formulations using the traveling salesman problem". In: *SIAM review* 45.1 (2003), pp. 116–123.

[27] G. Pesant et al. "An exact constraint logic programming algorithm for the traveling salesman problem with time windows". In: *Transportation Science* 32.1 (1998), pp. 12–29.

[28] M. Posti, J. Schöning, and J. Häkkilä. "Unexpected journeys with the HOBBIT: the design and evaluation of an asocial hiking app". In: *Proceedings of the 2014 conference on Designing interactive systems*. ACM. 2014, pp. 637–646.

[29] D. Quercia, R. Schifanella, and L. M. Aiello. "The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city". In: *Proceedings of the 25th ACM conference on Hypertext and social media*. ACM. 2014, pp. 116–125.

[30] M. Raifer. *Overpass turbo/Polygon Features*. Apr. 30, 2016. URL: http://wiki.openstreetmap.org/wiki/Overpass_turbo/Polygon_Features.

[31] C. Rego et al. "Traveling salesman problem heuristics: Leading methods, implementations and latest advances". In: *European Journal of Operational Research* 211.3 (2011), pp. 427–441. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/j.ejor.2010.09.010.

[32] J. Robinson. *On the Hamiltonian game (a traveling salesman problem)*. Tech. rep. DTIC Document, 1949.

[33] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier Science, 2006. ISBN: 9780080463803.

[34] N. Runge et al. "No more Autobahn!: Scenic Route Generation Using Googles Street View". In: *Proceedings of the 21st International Conference on Intelligent User Interfaces*. ACM. 2016, pp. 147–151.

[35] P. Shaw. "Using constraint programming and local search methods to solve vehicle routing problems". In: *Principles and Practice of Constraint Programming—CP98*. Springer, 1998, pp. 417–431.

[36] G. Skoumas et al. "Knowledge-enriched route computation". In: *Advances in Spatial and Temporal Databases*. Springer, 2015, pp. 157–176.

[37] W. Souffriau et al. "The planning of cycle trips in the province of East Flanders". In: *Omega* 39.2 (2011), pp. 209–213.

[38] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. "The orienteering problem: A survey". In: *European Journal of Operational Research* 209.1 (2011), pp. 1–10.

[39] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf. "An extension of the arc orienteering problem and its application to cycle trip planning". In: *Transportation research part E: logistics and transportation review* 68 (2014), pp. 64–78.

[40] D. B. West. *Introduction to graph theory*. Ed. by P. Education. 2nd ed. 2002.

[41] *WikiLoc.com*. URL: `http://www.wikiloc.com/wikiloc/home.do`.

[42] J. Zhang, H. Kawasaki, and Y. Kawai. "A tourist route search system based on web information and the visibility of scenic sights". In: *Universal Communication, 2008. ISUC'08. Second International Symposium on*. IEEE. 2008, pp. 154–161.

[43] Y.-T. Zheng et al. "GPSView: A scenic driving route planner". In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9.1 (2013), p. 3.

Appendix A

Pleasant values assessment : Resulting maps

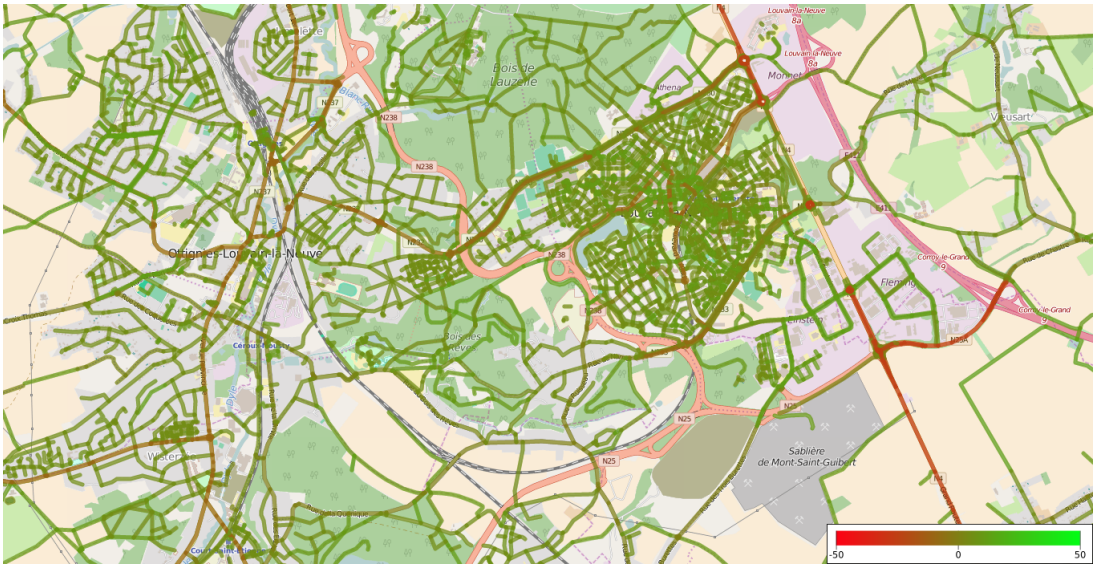


Figure A.1: Map representing the R part of the pleasant value for the `foot` mode ($\alpha = 0$)

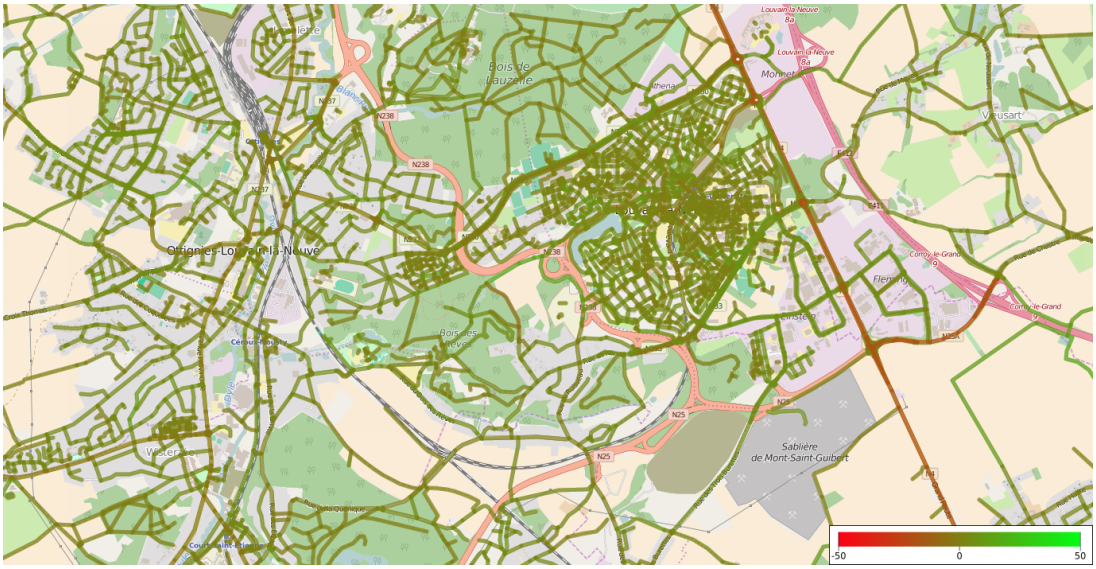


Figure A.2: Map representing the R part of the pleasant value for the `bicycle` mode ($\alpha = 0$)

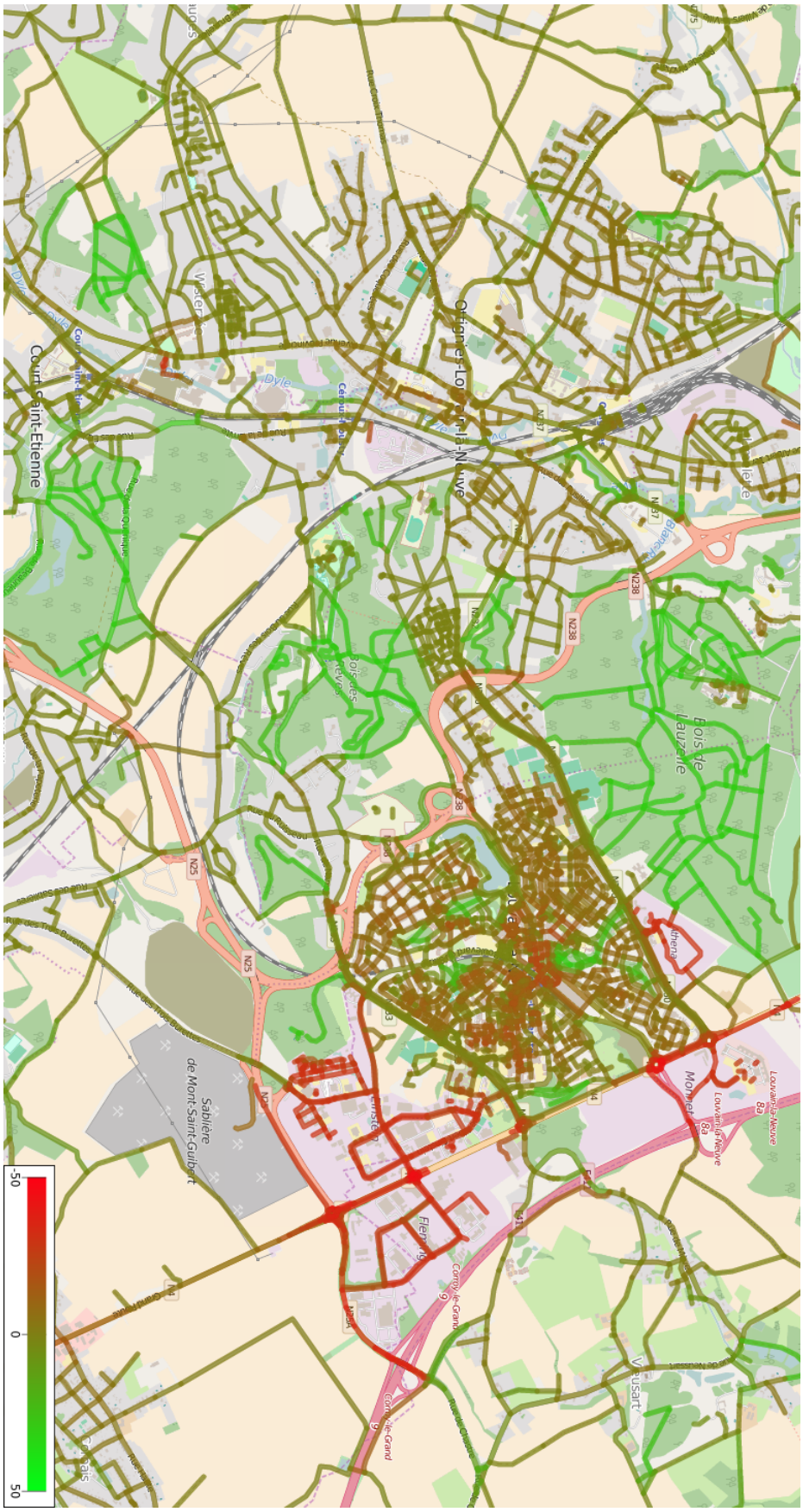


Figure A.3: Map of Ottignies-Louvain-la-Neuve representing the E part of the pleasant value for the `forest` predefined environment profile ($\alpha = 1$).

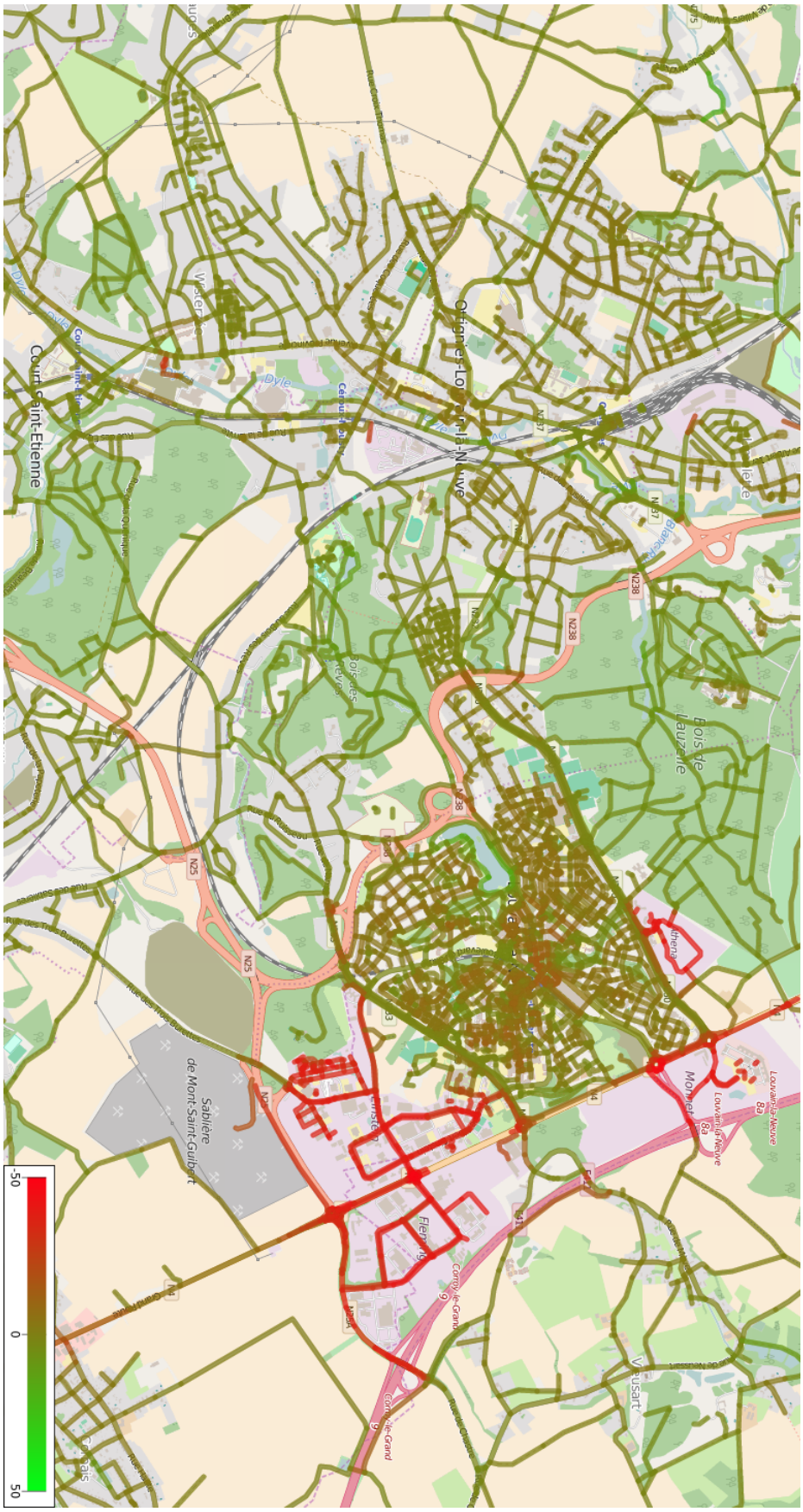


Figure A.4: Map of Ottignies-Louvain-la-Neuve representing the E part of the pleasant value for the `water` predefined environment profile ($\alpha = 1$).

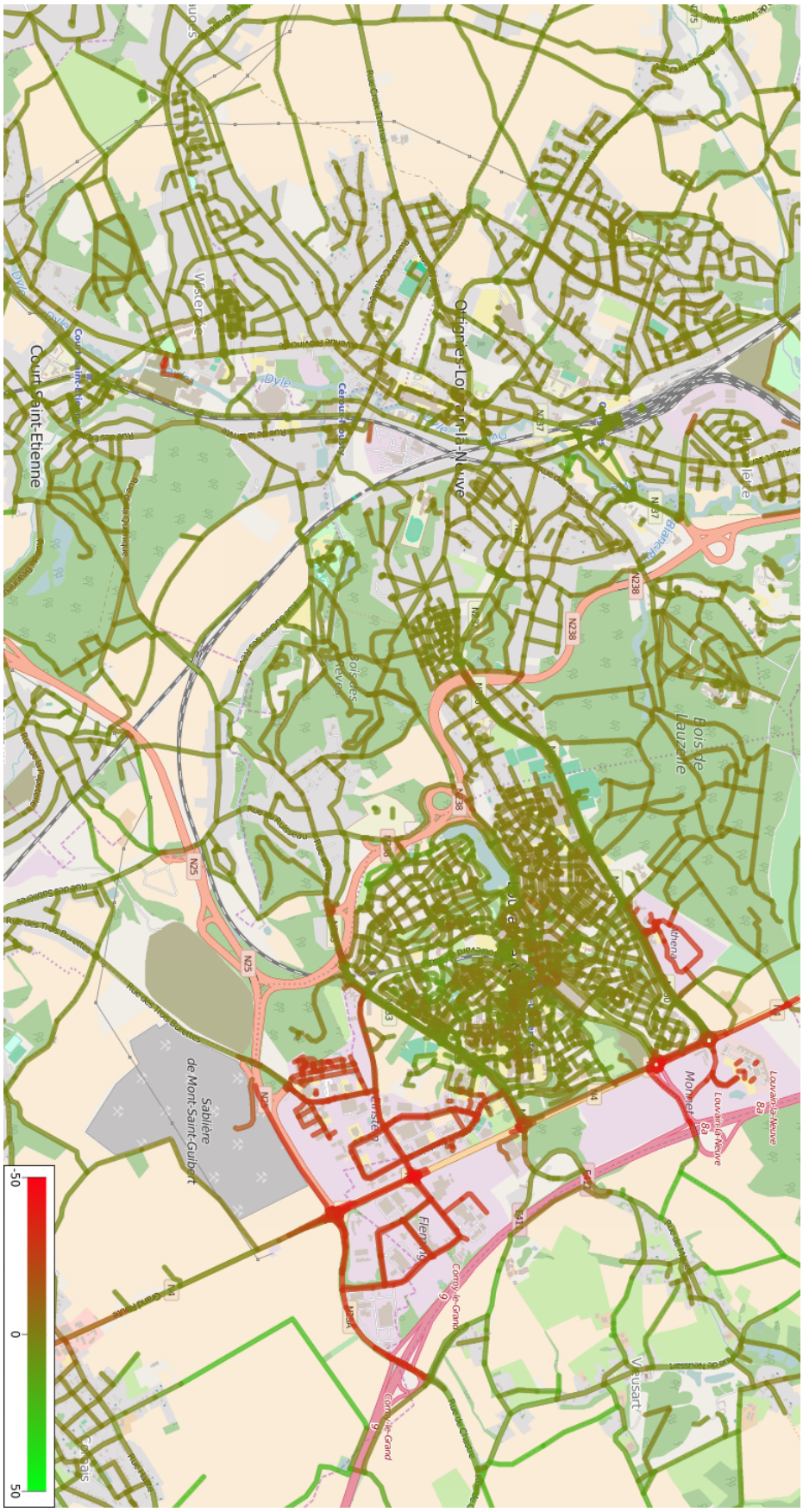


Figure A.5: Map of Ottignies-Louvain-la-Neuve representing the E part of the pleasant value for the `fields` predefined environment profile ($\alpha = 1$).

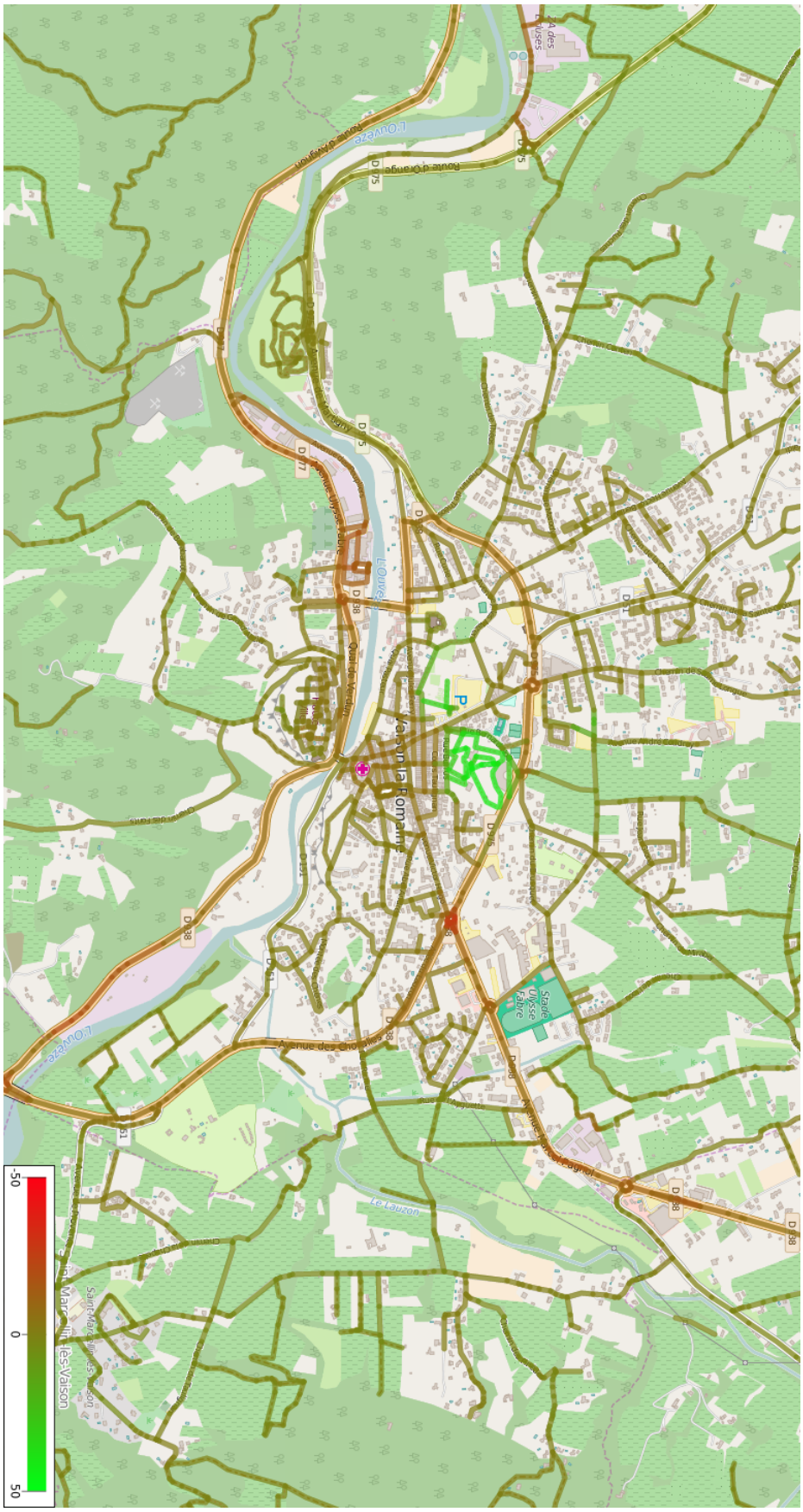


Figure A.6: Map of Vaison-la-Romaine representing the E part of the pleasant value for the `historic` predefined environment profile ($\alpha = 1$).

Appendix B

Quantitative data about the code

The table below presents some quantitative data about the code sorted by language used. For the Javascript and the CSS, we don't count the library used nor the minified version of the files. The entire source code is joined to this master as an archive file and constitute the third appendix.

Languages	**Number of files**	**Total size of files**	**Total number of lines**
Scala	37	157Ko	4468
PostgreSql / SQL	10	34Ko	869
HTML	2	23Ko	443
Javascript	1	28Ko	862
CSS	1	2.8Ko	122

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**