Free University of Bozen/Bolzano
Faculty of Computer Science

A Heuristic Itinerary Planning Approach for Objects With Time Constraints

Gytis Tumas

Thesis supervisor: Prof. Johann Gamper

Submitted in partial fulfillment of the requirements for degree of Master in Computer Science at the Free University of Bozen-Bolzano

October, 2011

Acknowledgments

I am grateful for all the people who assisted me in writing the thesis.

First of all I want to thank prof. Johann Gamper for the supervision, support and advices throughout my work.

Thanks to Andrea Janes for the constructive feedback and useful hints.

Last but not least I want to thank the Central Tourist Office of Bolzano-Bozen, and especially the Info Team member Nadja Sternkopf, for the availability and provided information, which was helpful for my dissertation.

Abstract

This thesis deals with a context- and user-aware approach to the itinerary planning in Geographic Information Systems (GIS). In order to assist tourists in planning and enjoying their trips we present the Orienteering Problem with Multiple Time Windows (OPMTW), a new routing problem combining objectives and constraints of the existing routing and scheduling problems such as Orienteering Problem, Vehicle Routing Problem and Travelling Salesman Problem with Time Windows. An effective iterative local search heuristic method is proposed to solve the OPMTW in real time and it is implemented for the client-server architecture with an application for mobile devices running the iOS operating system.

In the course of this work, existing problems of routing and itinerary planning are discussed and some of the most popular existing commercial itinerary planning tools are analysed. The obtained results from the computational tests confirm the effectiveness and the fastness of the introduced heuristic approach for the itinerary planning and suggest that that the full-scale application for tourists is feasible.

Contents

List of Figures

List of Tables

1 Introduction

Sometimes getting lost in a city is the best way to know it. However travellers more often prefer to be aware of their location and have the knowledge of different places of the city beforehand in order to choose the way how and when to visit them. On the other hand planning a good itinerary in an unknown city is not an easy task for a person to solve by herself. Even having the access to the data about the city and its famous objects, the quantity of the information can be overwhelming to make the right decision. There are usually plenty of factors for a user to consider, such as the types of the different points of interests, their reachability, the opening hours, entrance prices, distances between each other, etc. Besides that people who wish to explore the new city are typically interested not only in the famous objects of the city but also in the events happening there, such as concerts or exhibitions. Moreover the experience of the visit to a city may be often enriched by dining at the local places, going out for a drink or simply a snack.

In most cases an average day-tripper tourist plans his trip by consulting the websites, forums or the printed guidebooks. Since the time span of a trip is usually too short to visit all places of interest (POIs), a user chooses some places that she believes will be the most valuable. Once the decision is made, a tourist has to decide at what times the POIs could be visited and starts planning the route. However as noticed by Vansteenwegen et al. [1], the tourist comes across several problems by choosing an approach like this. First of all the information in the guide books is rarely up-to-date and the opening hours of the different POIs can change frequently: the theater plays can be moved and museums can have renovations and be closed for some time. Morover even if a tourist is able to identify the most valuable attractions and construct the route to visit all of them while staying within the schedule, she remains unaware if the route is optimal and if there are other routes with the better schedules available.

Another option for a tourist is to delegate the itinerary planning task to the staff of the Local Tourist Organizations (LTOs). LTOs usually combine the information obtained from the tourists (i.e. their preferences, time and money constraints) with the knowledge about the local attractions (type, opening hours, location) and suggest an itinerary for the tourist. However one of the problems that LTOs face is the speed of suggesting an itinerary: sometimes it is impossible to consider all the information about the user and the knowledge about the POIs in order to suggest an itinerary in a feasible time, thus the LTOs limit themselves with suggesting some predefined itineraries in this way partially or completely ignoring the user profile. Another problem that LTOs face are the legal constraints, because of which the businesses such as restaurants cannot be recommended explicitly in order to main the LTO's neutrality.

In order to address the problem of the itinerary planning for tourists when the opening times of the places are important, existing similar problems have been investigated, including the different variations of the well-known Travelling Salesman Problem (TSP). Additionally, some of the most popular existing and publically available commercial web-based itinerary planning tools have been analysed. Based on the empirical evaluation of the commercial tools and the analysis of the problems that address the construction of itineraries, we define our own variant of the Orienteering Problem with Time Windows (OPTW) which takes into account multiple time windows and construct a fast and effective iterative local search heuristic method to solve it. Moreover the developed algorithm has been implemented for the client-server architecture together with an application for mobile devices with the iOS operating system, that is able to generate multiple itineraries in a feasible time for a user. Computational tests confirm the effectiveness and speed of

the methods used, and suggest that the full-scale itinerary planning application is feasible. The developed system is also tested by an expert in a tourism field and is evaluated qualitatively.

2 State of the Art

2.1 Existing Tools for Itinerary Planning

To address the problem and help a user not to get lost in the tremendous amount of information while travelling, different applications and systems have been developed in the recent years, that try to construct the itineraries based on the user feedback of various granularity. In this part some of the most recent online tools that perform this task are analysed.

2.1.1 YourTour

YourTour is a website that lets users create multiple-day road trip itineraries. As the first input it requires only the departure/destination points (that can be selected either as a city or an airport), travelling dates and the amount of people travelling. Given this data the tool constructs the detailed customized path connecting different places, selects the hotels to stay and provides the estimated fuel costs required to traverse the path (Figure 1). After that a user can customize the suggestions by specifying the amount of her interest for the different categories (Culture, Shopping, Nature, etc.) as well as the preferences for the hotels.

The *YourTour* provides the detailed information about the places of the tour, such as the opening hours and the admission prices. However while focusing on the inter-city travellers with their own transport means the service unfortunately ignores the users who choose the sightseeing of a particular city on foot - the street level data is not present. Besides that the service fails to provide restaurant recommendations and makes it difficult to view the suggested itinerary on a mobile device.

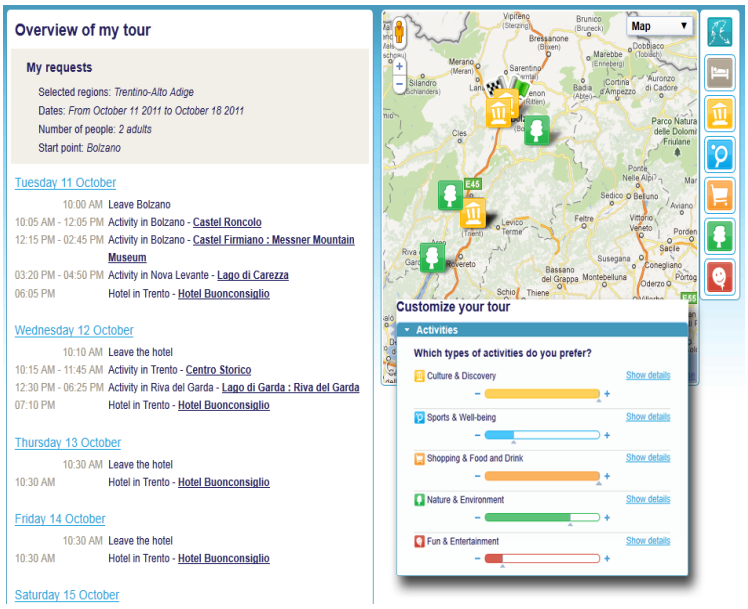


Figure 1: YourTour suggested path and its customizable preferences

2.1.2 Plnnr

Plnnr is another web based itinerary planner. Different from *YourTour*, the *Plnnr* is focused on the multiple-day tours within one of the 20 world's famous city. Additional to the typical input parameters this services takes into account such preferences of a user as the preferred trip type (more outdoors, culture, a trip with kids) and the intensity of the trip. For each day a different itinerary is generated, which always starts and ends at the user's hotel and can be further customized by changing the initial preferences or simply tweaked by removing unwanted places and adding desirable places from the categorized lists into the already generated itinerary.

Like the *YourTour* tool, the *Plnnr* service also takes into account the opening hours of the places and the time it takes to travel between them on foot or by taxi. However while the service proposes the daytrips filled with the schedules of visiting different places in a city, it both lacks the functionality to include the places to have a lunch or dinner and does not allow a user to plan it herself. Besides that, like *YourTour*, the *Plnnr* is not optimized for a mobile device, which makes it hard to customize the trip while being on the move.

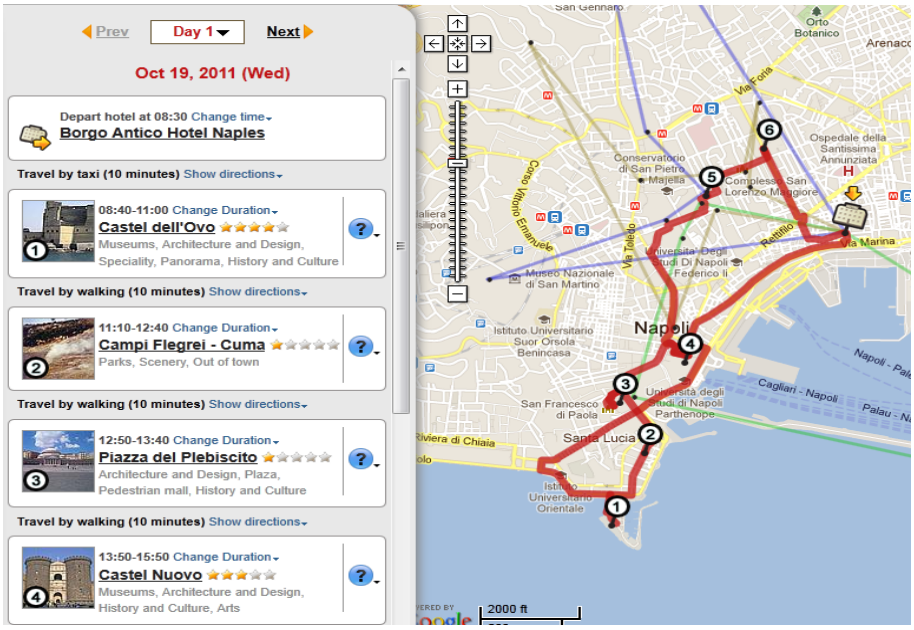


Figure 2: Screenshot of the Plnnr tool showing a generated itinerary

2.1.3 GoPlanIt

Similar to *Plnnr*, *GoPlanIt* is focussing on single or multiple day trips around a city, supplying the schedules of POIs to be visited and maps for the route display. The suggestion is mostly based on the preferences specified by the user and the proximity of the POIs. After the trip has been generated a user can always modify it by adding, deleting or rearranging the items, in the calendar-style graphical interface (Figure 3). If the opening hours of the POIs are flexible enough, the user can expand or decrease the time slot assigned by the tool for a specific POI. *GoPlanIt* has the cleanest interface compared to the previously mentioned application and it allows several views of the itinerary: using

the calendar view a user can analyse better the schedulling proposals and modify them, using the map view a user can visualise the categorized POI and see their proximity to each other, while using the printout view the user gets all available information about a specific itinerary in the print-out friendly format. The main differentce between *GoPlanIt* and the previously mentioned tools is that *GoPlanIt* smartly integrates the restaurant recommendations based on user's budget. Moreover it is possible to access the information of every suggested POI in details, print out the route with its information or simply access it on a mobile device. The main drawback of the tool is its data availability: so far the itineraries can be planned only for the biggest US cities.

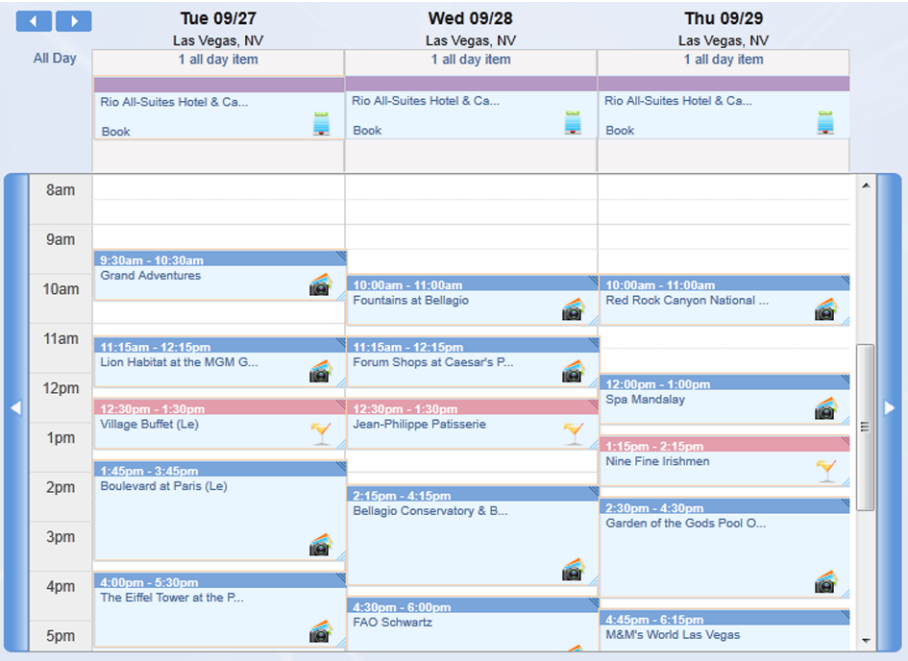


Figure 3: The calendar-style recommendation made by the GoPlanIt web tool

2.2 Time Independent Approaches for the Generation of Routes

Many different studies have been carried out analysing the ways to generate the routes by solving the Travelling Salesman Problem (TSP), the Prize-Collecting Traveling Salesman Problem (PCTSP) or the Orienteering Tour Problem (OTP). Such problems as the PCTSP and OTP are generally called the Travelling Salesman Subtour Problems (TSSP) [2]. Another similar family of problems are the Vehicle Routing Problems (VRP). All algorithms available for finding optimal solutions both for TSSP problems and the VRPs have the exponential complexity with respect to the size of the problem [17]. Therefore most of the studies try to find some heuristics, which usually allows to obtain reasonable solutions even though they cannot guarantee the optimal solution. In the following subsection first the classical Travelling Salesman Problem (TSP) will be described and then we take a look at the subproblems and how they can be applied for the generation of the routes.

2.2.1 Travelling Salesman Problem

The TSP is one the most studied problems in the combinatorial optimization. Because of the problem's simplicity and its applicability in different fields (tourism, transport logistics, material handling in a warehouse), plenty of exact and heuristic optimization algorithms had been developed for it ([3], [4], [5], [6]). The main idea behind the problem is the salesman who must travel to a set of cities. The task for the salesman is that of finding the shortest possible tour between all of the cities visiting each one of them exactly once. The pairwise distance between the cities is given. The problem may be formally defined as follows. We are given a graph $G = (V, A)$, where $V = \{v_1, ..., v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V\}$ is the arc set. A non-negative cost c_{ij} is assigned for each arc. The problem is to find the least cost circuit passing through all the vertices visiting every vertex exactly once. Such a circuit is called Hamiltonian circuit.

The TSP can be modelled as an undirected weighted graph, such that the cities are the vertices of the graph, the paths are the edges and a path's distance is the edge length. Usually every pair of vertices is connected by an edge, therefore the graph is complete. The Travelling Salesman Problem can be either symmetric or asymmetric. When we consider the symmetric TSP the distance between two cities does not depend on the direction with which we traverse the edge. Thereforethe symmetric TSP is modelled with an undirected graph. On the other hand in the asymmetric TSP paths migh not exist for both directions between two cities or the distance can be different. This might happen if we consider one-way streets or the up-hill streets, where it takes longer time to travel up-hill than going back down-hill.

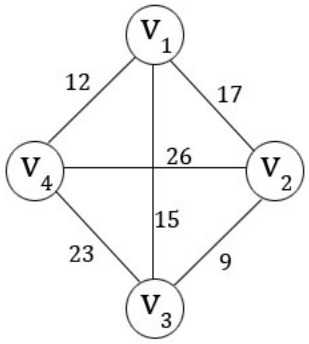


Figure 4: A symmetric Travelling Salesman Problem with four cities

2.2.2 Variants of the Travelling Salesman Problem

Prize-Collecting Travelling Salesman Problem (PCTSP), first mentioned by Balas et al. [7], is a relaxed version of the TSP. While in the original problem a salesman has to visit all the cities and his goal is to minimize the overall cost of the route, in the PCTSP a salesman does not necessarily need to visit all the cities, but must collect a given quota of weights that are associated to the cities (so called prizes). The total cost will be the length of the trip added to the penalties of the cities that were not visited by the salesman. Quota Traveling Salesman Problem is a special case of the PCTSP where all penalties are equal to zero [8].

Orienteering Problem (OP) is another extension of the Travelling Salesman Problem, first introduced by Tsiligrides [10]. It is also known as the Selective Travelling Salesman

Problem or the Maximum Collection Problem. Similarly to the PCTSP, the Orienteering problem does not seek to include all of the cities/nodes, but rather tries to construct a path with the maximal total score given the maximal duration of the path as a constraint. One application scenario for the Orienteering Problem suggested by [18] is a company, that must services a set of customers, each one with associated different profit. Limited number of resources (e.g. given time) restrict the number of customers that can be served and thus force the company to choose just a subset of them, preferably the ones with the higher profit.

Joest et al. [9] introduce the **Enhanced Profitable Tour Problem (EPTP)**, which is a variation of the Orienteering Problem (OP). Similarly like the previously defined approach, the problem consists in finding a particular cycle that maximizes the sum of the prizes visiting each node at most once and not exceeding the given total cost value t_{max}. The EPTP differs from the OP by having the weight assigned both to the nodes and to the arcs of the graph.

The Vehicle Routing Problem (VRP) can be described as the problem of designing optimal delivery as the collection routes from one or several depots to a number of geographically scattered cities or customers, subject to side constraints [11]. It is commonly used in practice in scenarios where a set of vehicles has to distribute the goods from depots to a set of customers. The problem may be formally defined as follows. We are given a graph $G = (V, A)$, where $V = \{v_1, ..., v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V\}$ is the arc set. After a non-negative cost c_{ij} is assigned for each arc, the VRP consists of finding a set of least cost routes visiting every vertex exactly once by exactly one vehicle, where each vehicle starts from a speficied point, known as depot.

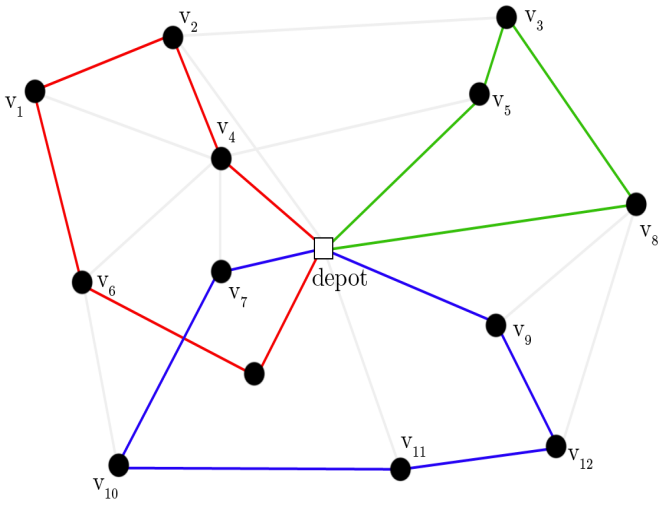


Figure 5: Vehicle routing problem

There are different variations of the Vehicle Routing Problem itself, which usually differ by imposing one or several side constraints to the original problem. One family of the VRP subproblems are the **Capacity constrained VRPs (CVRPs)**. The capacity is restricted by assigning a non-negative weight to each point different from the depot. Then the sum of weights of any rout cannot exceed the given vehicle capacity.

Another constraint that is commonly imposed on the Vehicle Routing Problem is the bounding of number of cities on any generated route above some value k. Given this

constraint a problem can be modelled as a special case of the previously described CVRP when each node is assigned a weight of 1 and the vehicle capacity is k.

2.3 Time Dependent Approaches for the Generation of Routes

When generating a tour for travellers the path can be constructed by solving one of the previously described problems. The choice of the problem would depend on the constraints that we want to impose for the travellers and how we want to design the graph (for example, if we want to have nodes with different weights/prizes or we allow the user to specify the maximal cost/length of the path). Our problem of the itinerary generation could be successfully mapped to one the problems described before, only if we ignored the time constraints, i.e. any node can be visited at any time. However in order to better represent the real life scenarios the time constraints of the different points must be considered. With the following example it is shown that modelling the path construction as the classical Travelling Salesman Problem is impractical for our scenario.

Consider a tourist who wishes to start his itinerary from his hotel at 10am in the morning and finish the trip at 4pm. The graph contains the nodes that correspond to the set of the potential points of interest (POIs) that are assumed to be of great interest for a user (See Figure 6). The nodes r, m, s and h represent a restaurant, a museum, a shopping center and a hotel respectively. The hotel is the start and the end point of the trip. For each object n we are given its opening hours (n_{open}) and the average visiting time (n_{visit}). The weights of arcs represent the distances in minutes between different nodes.

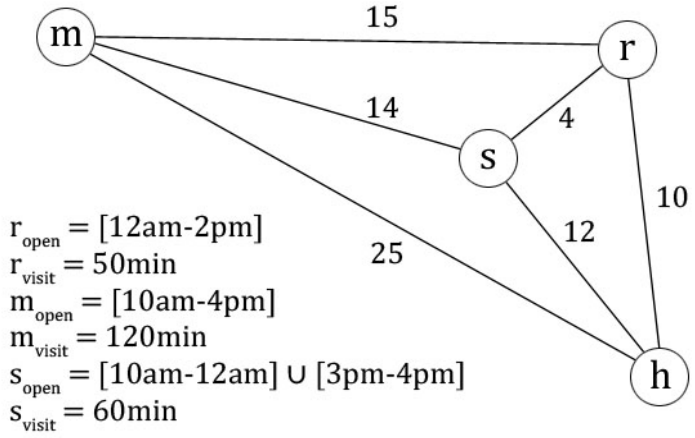


Figure 6: A sample graph with different POIs and the data about them

Trying to solve the Travelling Salesman Problem on the given graph gives us the solutions $S_1 = (h, r, m, s, h)$ and $S_2 = (h, s, m, r, h)$ that are in fact the least cost circuits passing through all the vertices. However despite the optimal travel distance both of the solutions would be infeasible in the real life scenario because of the given time constraints. For instance, using the path S_2 a tourist would start his trip at 10am from his hotel (h). From 10:12 till 11:12 he would be at the shopping center (s), still satisfying the opening hours constraint. Based to the given data, the user should be at the museum (m) for the time period between 11:26 and 13:26. After that even if the user is scheduled to arrive at the restaurant at 13:41, which is still within the opening hours constraint, the visiting of the restaurant would be not feasible anymore because of the given average visiting interval

for r, which is 50 minutes.

Instead of skipping some of the objects, the smarter solution would be to suggest the path $S_3 = (h, m, r, s, h)$, following which the user would start her trip at 10:00, reach the museum (m) at 10:25, spend there 2 hours, then after 15min of walking arrive to the restaurant (r) at 12:40. Since the restaurant (r) is opened from the noon till 2pm, the user would be able to fit within the time window, leaving the place at 13:30. From the restaurant (r) he would arrive at the shopping center at 13:34, leave it at 14:34 and finish the trip in the hotel (h) at 14:46. Even though the travel cost to traverse the S_3 path is higher than S_1 and S_2, the path S_3 is much more suitable for a user in the real life scenario.

In order to take into account the opening hours of the objects, we must consider the constraint that restricts the objects in a graph to be visited outside of their opening hours (time windows). I describe several of the problems that consider this constraint.

2.3.1 Travelling Salesman Problem with Time Windows

The Traveling Salesman Problem with Time Windows (TSPTW) is a time constrained variant of the TSP. The problem is NP-hard, and Savelsberg [13] showed that even finding a feasible solution of TSPTW is NP-complete. In the TSPTW the goal is to find the minimal cost path visiting all the cities exactly once and at the same time respecting their time windows constraints, i.e. visiting every node only within its give time window. The early arrival times are allowed, so given the time window $[a_i, b_i]$ for a node i, a salesman can be at the node i before its opening time a_i, however in this case the salesman must wait for the opening time. Moreover every vertex has its service time, i.e. how long a salesman must wait in the node (starting at least from the opening time a_i) before proceeding to visit the other vertices.

TSPWT is formally defined as follows. We are given a complete undirected graph G=(V,A) where $V = \{v_0, v_1, ..., v_n, v_{n+1}\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in A, i \neq j\}$ is the arc set. Vertex v_0 is the starting point, and vertext v_{n+1} is the end point. Each vertex must be visited during its time window $[a_i, b_i]$. The aim is to determine a directed path starting at v_0, ending at v_{n+1} and passing through all vertices in V exactly once. For the minimization objective, there exist two variations of the problem: in the first the objective is to minimize the arrival time at the vertex v_{n+1}, indepently of the path costs, while in the second varation of the TSPWT the objective is to minimize the overall cost, i.e. the travel time on the path, not taking into account the time it takes to wait at each vertex. The second objective is considered more often in the literature.

There are many heuristics and exact algorithms developed for the Travelling Salesman Problem with Time Windows. López-Ibáñez et al.[12] have developed a heuristic Beam-ACO approach, which is a hybrid method combining ant colony optimization with beam search. Using it the authors were able to obtain optimal or near-optimal (less than 1% deviation) solutions in a reasonable computation time. Focacci et al. [14] provide an exact algorithm to solve the TSPTW, by introducing a new approach which merges Constraint Programming propagation algorithms for searching for the path and Operations Research techniques for finding the best path.

2.3.2 Orienteering Problem with Time Windows

Orienteering Problem with Time Windows (OPTW) differs from its "cousin" TSPTW in a similar way as Orienteering Problem (OP) differs from the TSP. Same like the OP, the Orienteering Problem with Time Windows does not seek to include all of the cities/nodes, but rather tries to construct a path with the maximal total score of the visited nodes given the maximal duration of the path T_{max} as a constraint. In the OPTW a weight, a time window and a service time are associated with every node and a cost is associated with every link. The main goal is to find a path maximizing the benefit for it (which is the sum of the weights of visited nodes), where every node in the graph is either visited within its time window or skipped. The problem is classified as NP-hard problem. Righini et al. [15] present an exact optimization algorithm for the OPTW based on dynamic programming with decremental state space relaxation strategies. Morover Righini et al. [15] also propose a heuristic approach for the OPTW problem, where the main focus is on the initialization of the critical vertex set. It is shown that the new technique reduces the number of iterations and the computing time needed by the decremental state space relaxation strategies (DSSR) algorithm to converge to an optimal solution.

2.3.3 Team Orienteering Problem with Time Windows

The Team Orienteering Problem with Time Windows (TOPTW) can be viewed as the earlier described OPTW where the goal is to determine not one but m tours, each limited by the same given time budget T_{max}. The TOPTW can be also considered as a special case of the Vehicle Routing Problem with Time Windows and Profits. The main difference between the OPTW and the TOPTW is that in the later there is more than one route being created at the same time. An example application scenario could be the company, who wants to maximize the received profit, by sending a set of salesmen to visit a set of customers with different profits, where each customer can be visited just during his given time window.

Vansteenwegen et al. [16] propose a fast heuristic to solve the Team Orienteering Problem with Time Windows. The heuristic involves an insertion step and a shaking step to escape from local optima. In order to skip the feasibility checking for all visits when inserting a new node, the *Wait* and *MaxShift* values are recorded. *Wait* is defined as the waiting time at a vertex v_i in case the arrival to the node v_i takes place before its opening time a_i. *MaxShift* is defined as the maximum time the service completion of a gine visit can be delayed. In Section 6 we are going to compare the heuristic developed by Vansteenwegen et al. and our proposed solution.

3 The Problem Definition and Heuristics

In this chapter the Orienteering Problem with Multiple Time Windows (OPMTW) is formalized, which will be used to model and solve our itinerary planning problem for tourists. Later we propose the fast and effective heuristic approach to solve the problem.

3.1 Orienteering Problem with Multiple Time Windows

We define the Oriented Problem with Multiple Time Windows (OPMTW) as follows. We are given a complete directed graph $G = (V, A)$, where $V = \{v_0, v_1, ..., v_n, v_{n+1}\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the arc set, where the vertex v_0 represents the starting point of the trip and the vertex v_n represents the end point of the trip. A non-negative cost c_{ij} is assigned to each arc, which represents the travel time from the vertex i till the vertex j for a specific transport mean. For each vertex $i \in V, i \neq 0, i \neq (n+1)$ we define the p_i as the profit of a vertex i, the k time windows $[a_{i1}, b_{i1}], ..., [a_{ik}, b_{ik}]$, where a_{ij} is the opening time of the jth time window for the vertex i, and the b_{ij} is the closing time of the jth time window and the s_i as the service time for the vertex i, which is the estimated time for a user to spend at the vertex i. For more details about the values for the average service time refer to the subsection 4.2.

For each pair of vertices (i, j) we define the service start time t_j as follows:

$$t_j = \begin{cases} d_j & \text{if } j = 0, \\ max(a_{jk}, t_i + c_{ij} + s_i) & \text{if } j \neq 0. \end{cases} \tag{1}$$

Here a_{jk} is the opening time of the earliest feasible time window of the node j, and d_j is the departure time from the node j.

In the OPMTW the aim is to determine the directed path P, such that it starts at the node v_0, passes through a subset of vertices of $V \backslash \{v_0, v_{n+1}\}$, visiting each node at most once, ends at the node v_{n+1}, maximizes the overall profit $\sum_{i \in P} p_i$ and respects the constraint $a_{i1} \leq t_i \leq b_{ik}$, where b_{ik} is the closing time of the latest time window of i.

3.1.1 Considering the Real World Features

Since the graph is directed it does not guarantee that $c_{ij} = c_{ji}$, which means that the travelling time from node i to the node j can be different when going in the opposite direction from j to i. This choice allows the better representation of the real city topologies, because the time to travel between two nodes/intersections can differ for both ways if a street is not flat when walking and the driving time between two nodes/intersections can differ if there are traffic congestions for one way and no for another.

In the real life the touristic objects usually have the opening hours that are composed of more than one time window of service per day. For example consider a restaurant which is opened for lunch from 12:00 till 15:00 and again in the evening for dinner from 18:00 till 23:00, or a shopping mall that has two time windows one in the morning and another in the afternoon, with the closing lunch break. In our problem definition the multiple time windows are considered, however the overlaps of the time windows for the same object are not allowed.

3.2 Heuristics

In this part the fast and effective heuristic approach that was constructed to solve the Orienteering Problem with Multiple Time Windows is described. The solution includes the modification of the insertion procedure proposed by Gendreau et al.[19]. This procedure was originally used as a step of the heuristics that solve the Travelling Salesman Problem where the path starts and ends at the same point. Later Gendreau et al. use the same procedure to solve the Travelling Salesman Problem With Time Windows [20]. The procedure has been modified in a way that it would be applicable for the Orienteering Problem with Multiple Time Windows and the start and the end points would not need necessarily to coincide.

We start the algorithm with any path consisting of the two vertices, one of them will be the start while the other - the end of the trip. At any iteration we try to insert a vertex v on the partially constructed path P while performing the local reoptimization. The insertion of a node v is attempted between two not necessarily consecutive nodes v_i and v_j, that are in $N_h(v)$, where $N_h(v)$ is the neighbourhood of the vertex v. The choice of the selection of the next node to insert into the path P is explained in the subsection 3.2.3. The neighbourhood $N_h(v)$ is the set of the closest h vertices already inserted in the path P, where h is the input parameter. In case when the number of vertices in the P is less than h, the neighbourhood $N_h(v)$ includes all of the vertices in P. It should be noted, that to find the closest vertices for a given vertex, we are not using the actual cost or the distance between them but rather the pseudo-distance, which is composed based on several factors and that takes into account also the time windows of a node. The definition of the pseudo-distance is explained in the subsection 3.2.2.

After each trial of insertion we check the feasibility of the path, so that the newly inserted vertex does not violate the multiple time windows constraint. For all possibilities of v_i and v_j in in $N_h(v)$, the best feasible insertion is implemented to continue constructing the partial solution for the path P. If there is no feasibile insertion for a v, we ignore it. We continue to procedure of the insertion for other vertices. The algorithm terminates either after the path P contains z vertices, where z is the input parameter, or all the vertices have been attempted for the insertion.

3.2.1 Selection of the Vertices

The selection of the vertices to insert is not performed at random, but it is based on the two criterions c_1 and c_2, where c_1 is "select the most profitable vertex so far for the user" and c_2 is "decrease the difficulty of the insertion of a new node". The choice of the c_1 is intuitive as we want to maxime the overall profit $\sum_{i \in P} p_i$ of the resulting path, thus we prefer to start inserting the vertices with higher profit. On the other hand, there are serveral candidate options for the c_2 criterion. The level of difficulty for the insertion of a node can be defined in one the following ways: increasing order of the sum of the vertices windows widths, increasing order of the widest time window of a vertex, increasing order of the minimal opening value for a time window a_i. Based on the results of the preliminary tests we choose the increasing order of the sum of the vertices windows widths, and use it as the c_2 criterion to represent the difficulty level of the insertion for the nodes in the partially constructed path P.

3.2.2 Constructing the Neighbourhood

As mentioned before, we try to insert a new vert v between two nodes that belong to the neighbourhood $N_h(v)$ of the the vertex v. The neighbourhood $N_h(v)$ consists of h closest vertices to v, where h is the input parameter. In the heuristic methods that have been developed for the problems without time windows, such as the classical Travelling Salesman Problem, it is convenient to use the actual cost between vertices when constructing the neighbourhood, as it was done in [19]. However when dealing with the approaches that take into account the time windows, the search for the closest vetices cannot be based just on the cost alone, but the distances between the time windows of different nodes have to be considered too.

Therefore we define the proximity of the time windows between two vertices v_i and v_j as follows:

$$r_{ij} = dt - ot_{[A_i,B_i],[A_j,B_j]} \tag{2}$$

where dt is the total time of a day, $ot_{x,y}$ - overlapping time of the time intervals x and y, $[A_i, B_i]$ and $[A_j, B_j]$ - the sum of the time windows for the vertex i and for the vertex j accordingly. The example of finding the proximity between two nodes is displayed in the figure 7.

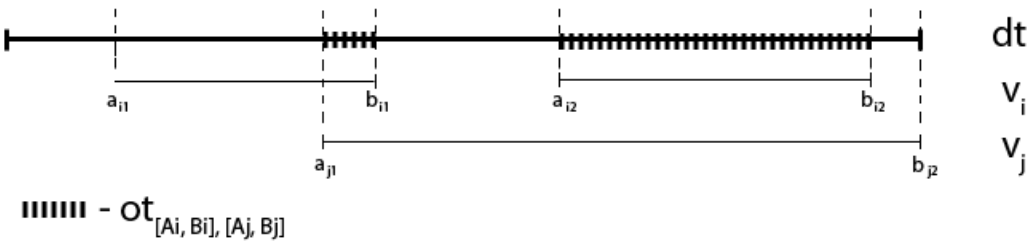


Figure 7: Finding the proximity between the time windows of two vertices v_i and v_j

One way to define the pseudo-distance d_{ij} would be to use a convex combination of the cost c_{ij} and the proximity r_{ij}:

$$d_{ij} = \alpha c_{ij} + (1 - \alpha) r_{ij}, where \alpha \in [0, 1]. \tag{3}$$

However it was shown by [20] that this formulation results in an unstable outcome, because setting the α value can be difficult as the cost value tends to dominate in some cases, while the time windows tend to dominate in other cases. Therefore a more convenient approach would be to add to the neighbourhood $N_h(v)$ the h_1 closest points of v already in the path P with respect to c_{ij} and h_2 closest points of v already in the path P with respect to r_{ij}, where $h_1 + h_2 = h$.

3.2.3 The Insertion Attempt

When inserting a vertex v into the partly constructed path P we are not necessarily trying to insert it between the two nodes that are consecutive in the path. However after the insertion the node v become adjacent to the two nodes. Imagine that we want to insert

the node v between two vertices v_i and v_j that are already in the path P. Let v_k be a vertex on the path from v_j till the last vertex of P and let the v_{l-1} and v_{l+1} be respectively the predecessor and the successor for any vertex v_l in the path P.

To insert the vertex v in the partially constructed path P we perform the following modifications:

- Delete the edges (v_i, v_{i+1}), (v_j, v_{j+1}) and (v_k, v_{k+1}).
- Insert the edges (v_i, v), (v, v_j), (v_{i+1}, v_k) and (v_{j+1}, v_{k+1}).

Note that such restructuring of the path results in the reversal of the subpaths $(v_{i+1}, ..., v_j)$ and $(v_{j+1}, ..., v_k)$. The procedure of the insertion is depicted in the figure 8.

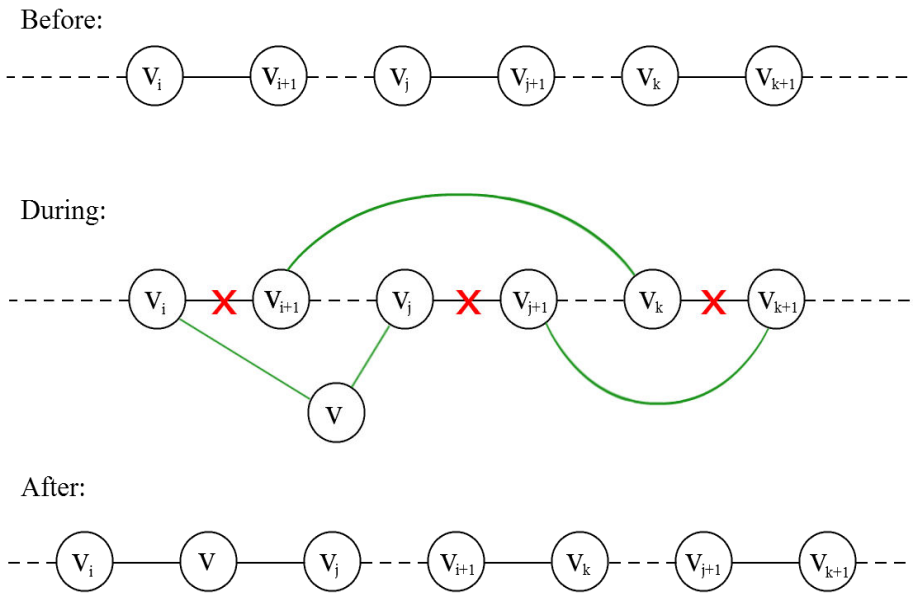


Figure 8: The subpath of P before, during and after the insertion of the vertex v

3.2.4 Summary of the Algorithm

The step by step summary of the algorithm:

Step 1 - Initialization

Insert all vertices of $V \backslash \{v_0, v_{n+1}\}$ into the set S. Order the set based on 2 parameters: first order decreasingly on the profit of the vertex, then if there are vertices that have the same profit sort them increasingly on the sum of the widths of their time windows. Start constructing the path P by adding the first two vertices v_0 and v_{n+1} to it. Initialize z as the parameter from preferences, defining the intensity of a trip, thus the amount of vertices within the path.

Step 2 - Insertion attempt

If $|P| = z$ or $S = \emptyset$ terminate the algorithm and return the constructed path P. Otherwise select the 1st vertex from the set S as v and attempt to insert it between two of its h

closest neighbours from its neighbourhood $N_h(v)$, using the insertion procedure described in 3.2.3.

Step 3 - Feasibility check

If at least on feasible insertion can be identified, implement the best one and update the path P. Remove the vertex v and go to Step 2.

3.3 Algorithm Illustration

The illustration of the algorithm is demonstrated on a sample graph consisting of 9 vertices $V = \{v_0, ..., v_8\}$, where v_0 is the starting vertex and v_8 is the ending vertex. The initial values for the parameters are the following: h=4, z=5, startingTime = 540, endTime = 1260, where h represents the size of the neighbourhood and z is the maximum number of vertices to be inserted. For every vertex v_i its time windows, the service time and the profit are specified in the table 9. Based on the profit values and the time windows the set S is initialized as follows: $S = \{v_3, v_1, v_2, v_6, v_5\}$.

vertex	time windows	service time	profit
V_1	[720;870], [1140;1350]	55	90
V_2	[510;720], [900;1170]	150	90
V_3	[0;1440]	90	100
V_4	[0;1440]	60	40
V_5	[540;1220]	180	50
V_6	[600;820], [880;1440]	15	75
V_7	[480;1440]	20	25

Figure 9: The initial data for the illustration of the heuristic approach for the OPMTW

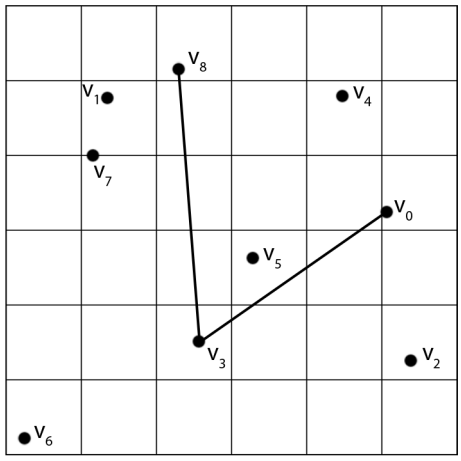
In every iteration the following information describing the current state is illustrated: the selected new vertex v_i for a potential insertion, the neighbourhood $N_h(v_i)$ of the vertex, the two vertices from the neighbourhood between which the vertex v_i was chosen for the best insertion, the contents of the path P, the arrival time s_{v_j} and the departure time d_{v_j} for every vertex $v_j \in P$, the travel cost and the overall profit of the current path.

The euclidean distances between the vertices in the planes correspond to the actual cost values between the vertices.

We start by putting the first two vertices into the path $P = v_0, v_8$.

3.3.1 Iteration 1

Selected vertex for insertion: v_3; $N_h(v_3) = P = \{v_0, v_8\}$; insert between v_0 and v_8; $P = \{v_0, v_3, v_8\}$



$s_{v_0} = \text{nill}$ $\quad d_{v_0} = 540$
$s_{v_3} = 570$ $\quad d_{v_3} = 660$
$s_{v_8} = 695$ $\quad d_{v_8} = \text{nill}$

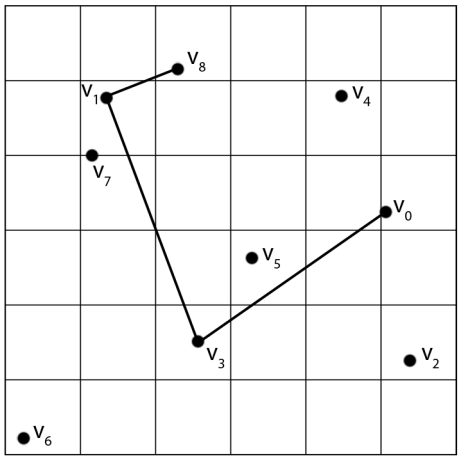
$\text{cost} = 65$
$\text{profit} = 100$

Figure 10: The graph data after the first insertion

For the first insertion the first vertex v_3 from the set S is chosen, as it has the highest profit value. Since there are only two vertices in the path P so far, the neighbourhood $N_h(v_3)$ includes both of them. The vertex v_3 is successfully inserted between the two initial vertices because its insertion does not break the time feasibility constraints.

3.3.2 Iteration 2

Selected vertex for insertion: v_1; $N_h(v_1) = P = \{v_0, v_3, v_8\}$; insert between v_3 and v_8; $P = \{v_0, v_3, v_1, v_8\}$



$s_{v_0} = \text{nill}$ $\quad d_{v_0} = 540$
$s_{v_3} = 570$ $\quad d_{v_3} = 660$
$s_{v_1} = 693$ $\quad d_{v_1} = 775$
$s_{v_8} = 785$ $\quad d_{v_8} = \text{nill}$

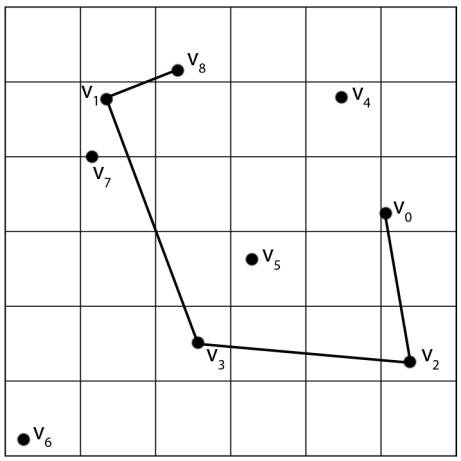
$\text{cost} = 73$
$\text{profit} = 190$

Figure 11: The graph data after the second insertion

For the second insertion the vertex v_1 is chosen. Even if the vertex v_2 has the same profit value as the vertex v_1, in this case the later has the priority because the sum of its time windows is smaller than the sum of v_2 time windows (for v_1 it is 360, for v_2 - 480). The neighbourhood of the vertex v_1 also still consists of all three vertices currently inserted into the path P.

3.3.3 Iteration 3

Selected vertex for insertion: v_2; $N_h(v_2) = P = \{v_0, v_3, v_1, v_8\}$; insert between v_0 and v_3; $P = \{v_0, v_2, v_3, v_1, v_8\}$



$s_{V_0} = \text{nill}$ $\quad d_{V_0} = 540$
$s_{V_2} = 560$ $\quad d_{V_2} = 710$
$s_{V_3} = 738$ $\quad d_{V_3} = 828$
$s_{V_1} = 861$ $\quad d_{V_1} = 1195$
$s_{V_8} = 1205$ $\quad d_{V_8} = \text{nill}$

$\text{cost} = 94$
$\text{profit} = 280$

Figure 12: The graph data after the third insertion

In the third iteration the vertex v_2 is selected as it has the biggest profit out of all remaining vertices in the set S. It is inserted between the vertices v_0 and v_3 as this insertion results in the cheapest new path P.

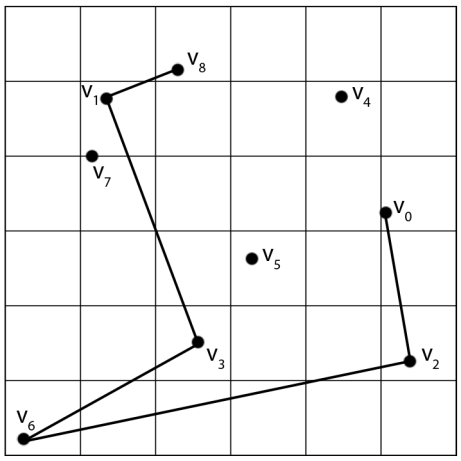
3.3.4 Iteration 4

Selected vertex for insertion: v_6; $N_h(v_6) = \{v_2, v_3, v_0, v_8\}$; insert between v_2 and v_3; $P = \{v_0, v_2, v_6, v_3, v_1, v_8\}$

The vertex v_6 is selected as the next potential candidate for the insertion. Because the given parameter h=4, for the first time the neighbourhood of a potential vertex does not correspond to the path P, but consist only of 4 vertices belonging to the path, between which the new vertex will be attempted for an insertion. The vertices v_2 and v_3 were added to the neighbourhood $N_h(v_6)$ for being the closest vertices to v_6 with respect to the cost c_{ij}, while the vertices v_0 and v_8 were added for being closest to v_6 with respect to the time windows proximity r_{ij}, defined in (2).

3.3.5 Iteration 5

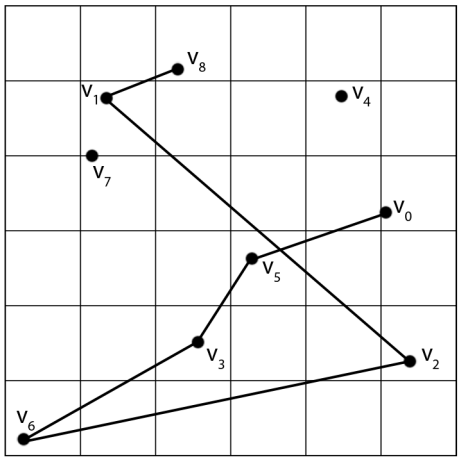
Selected vertex for insertion: v_5; $N_h(v_5) = \{v_0, v_3, v_1, v_8\}$; insert between v_0 and v_3; $P = \{v_0, v_5, v_3, v_6, v_2, v_1, v_8\}$



$s_{v_0} = \text{nill}$ $d_{v_0} = 540$
$s_{v_2} = 560$ $d_{v_2} = 710$
$s_{v_6} = 760$ $d_{v_6} = 775$
$s_{v_3} = 797$ $d_{v_3} = 887$
$s_{v_1} = 920$ $d_{v_1} = 1195$
$s_{v_8} = 1205$ $d_{v_8} = \text{nill}$

$\text{cost} = 140$
$\text{profit} = 355$

Figure 13: The graph data after the fourth insertion



$s_{v_0} = \text{nill}$ $d_{v_0} = 540$
$s_{v_5} = 555$ $d_{v_5} = 725$
$s_{v_3} = 735$ $d_{v_3} = 825$
$s_{v_6} = 847$ $d_{v_6} = 895$
$s_{v_2} = 945$ $d_{v_2} = 1095$
$s_{v_1} = 1145$ $d_{v_1} = 1200$
$s_{v_8} = 1210$ $d_{v_8} = \text{nill}$

$\text{cost} = 163$
$\text{profit} = 405$

Figure 14: The graph data after the fifth and final insertion

The vertex v_5 is attempted for the last insertion between its neighbours, which are the vertices v_0 and the v_3 for their cost c_{ij}, and the v_1 and v_8 for their time windows proximity. The best local insertion is determined to be between the vertices v_0 and v_3. Since v_0 and v_3 were not consecutive in the path P that is being constructed, the subpath v_3,...,v_0 will be reversed. This results in the fact that the vertex v_2 which belongs to the mentioned subpath will be visited on its second time window [900; 1170] rather than the first one [510; 720]. After the feasibility check of all time constraints, the path $P = \{v_0, v_5, v_3, v_6, v_2, v_1, v_8\}$ is returned as the output.

4 Application of the Heuristic Approach for Itinerary Planning

In this chapter we describe how we applied the proposed heuristic approach that solves the Orienteering Problem with Multiple Time Windows in order to make an itinerary planning system aimed for the visitors and tourists of South Tyrol. The main goal of the system is to suggest a day trip itinerary for a tourist within a city, which would satisfy the tourist most, given her time constraints and the preferences. Before designing the prototype some assumptions had to be made both about the users of the system and about their behavior. The validity of the assumptions have been evaluated together with the system itself by an expert using it. The results of the evaluation are described in section 6.

4.1 Sample Data

For the application a subset of the semi-actual data about of the most popular touristic places in Bolzano (Italy) were used. The descriptional data and the categorization of the points of interest (POIs) and the dining places have been downloaded from the free online worldwide travel guide WikiTravel (http://wikitravel.org/en/Main_Page). The website categorizes the points of interest into 4 main categories, namely "See", "Do", "Buy" and "Eat".

Each of the categories have different subcategories, such as "Musuems", "Churches", "Theaters" etc. Based on the categorization, and the selected subset of objects for our purposes of the system, 5 categories for the points of interest and 3 categories for the dining places were identified, namely:

- "Landmarks"
- "Museums & Galleries"
- "Entertainment & events
- "Nature & environment"
- "Pubs, bars, discos"
- "Budge dining places"
- "Mid-range dining places"
- "Splurge dining places"

Where available, the actual opening hours of the places were used, both for POIs and for dining places. Such objects as statues or parks were naturally assigned the 24/7 opening time. Apart from the type categorization, each object is also categorized as being an event or a simple point of interest. The events differ from the other objects first of all by the happening dates. While most of the objects have constant time windows for weekdays or weekends all over the year (except for holidays), the events usually happen only once in the life time or have a constant occurrence date, for example on a specific day every week. Thus the attention must be paid to the data selection preference of the user. Morover the restrictions of the visiting time of the place with respect to the time windows for

event objects is different than the one for other objects. Normally the non-event objects can be visited any time after the opening time of any of their windows, as long as the service time is not longer than the closing time of the window. On the other hand for the event objects the arrival of the user to the object after the opening time of the time window should not be feasible anymore. For example, we cannot suggest a tourist to visit a theater play, if he can arrive to the theater half an hour after the beginning of the play. However to address this issue we do not need to modify the algorithm. It is enough to set the service time of an event object equal to the the length of its time window. For example if a play in the theater starts at 8pm, ends at 10pm, the service time must be 120min. In this case the algorithm will consider any arrival after the beginning of the play as an infeasible vertex visit.

For the choice of the start/end points of the trip, the users are provided with the separate set of the objects, which are typically the most common starting places for the day-trip itineraries, containing the city hotels, hostels, several landmarks and the train/bus stations.

4.2 Estimating Values for the Average Service Time at Objects

When dealing with constructing itineraries for the touristic objects where the time constraints such as the opening times or the travel times between objects are of great importance, we cannot ignore another time constraint which is the service time per object. This value is highly subjective and is dependent on a big amount of factors, many of which are dependent greatly on the users themselves. What is more, in most of the cases the service time cannot be chosen as one of the user's preferences in advance, because a tourist is simply unaware of how much it would take to visit a specific gallery, have a lunch at a restaurant or enjoy a park.

However there are automated methods which allow to predict the average time a user would spend in a specific place. One way to automatically obtain the data of the average visiting times is by using the publicly available Flickr data [25]. Flickr is one of the most popular image hosting portals, which allow the users to access the geotagged data through its api. The tagging of the photos usually include not only the geo data, but also the POI names and the time information when the pictures were taken. Having access to this information, it is possible to predict the average staying time of the tourists in different places, assuming the constant frequence of the photo taking process. Since in our approach we are addressing also the places for dining, it is improbable that the frequence of taking photos while having lunch can be constant. We leave the analysis of the geotagged and timetagged photo data and its application for our approach for the future work and instead assume the heuristic average service values for different places.

4.3 Calculation of the Costs Between the Objects

In order to calculate the travel costs between the different objects, we generate the transit time between every pair of the object in the city by using the publicly available Google Directions API (http://code.google.com/apis/maps). The Google Directions API is part of the Google Maps API Web Services, which is a collection of HTTP interfaces to Google services providing geographic data. These web services use HTTP requests to specific URLs, passing URL parameters as arguments to the services.

Generally, these services return data in the HTTP request as either XML or JSON. The

Google Directions API calculates the distance as part of the directions query, when users specify origins, destinations and the optional waypoints. The data can be either specified as text strings (e.g. "Bolzano, Italy" or "Piazza Walther, Bolzano, Italy") or as latitude/longitude coordinates. For the higher precision we are using the latitude/longitude coordinates. The request data also contains the parameter that defines the transport mean (walking, driving, public transport). The sample JSON response from the Google Directions API is shown in the Figure 15.

We store every calculated pair-wise costs between all objects. It should be noted, that the travel cost between two objects o_i and o_j can be different than the one going backwards from o_j and o_i.

The sample request for the Google Directions API service:

http://maps.googleapis.com/maps/api/directions/json?origin=46.500031,11.349395&
destination=46.497483,11.349101&mode=walking&sensor=false&units=metric


```
"routes" : [
   {
      "bounds" : {
         "northeast" : {
            "lat" : 46.500050,
            "lng" : 11.34940
         },
         "southwest" : {
            "lat" : 46.49749000000001,
            "lng" : 11.349050
         }
      },
      "copyrights" : "Map data Â©2011 Tele Atlas",
      "legs" : [
         {
            "distance" : {
               "text" : "0.3 km",
               "value" : 292
            },
            "duration" : {
               "text" : "3 mins",
               "value" : 199
            },
            "end_address" : "Via Dante, 2, 39100 Bolzano Province of Bolzano-Bozen, Italy",
            "end_location" : {
               "lat" : 46.49749000000001,
               "lng" : 11.349120
            },
            "start_address" : "Via Museo, 54, 39100 Bolzano Province of Bolzano-Bozen, Italy",
            "start_location" : {
               "lat" : 46.50004000000001,
               "lng" : 11.34940
            },
```


Figure 15: The snippet of the sample JSON response to the Google Directions API query

4.4 User Profile

When initially modelling the user needs some of the assumptions about the potential users of the system had to be taken into account. Later on the assumptions were evaluated by an expert in the tourism field.

The following assumptions had been made about the potential users of the system:

- a_1 - a user prefers the itineraries that are shorter in distance, i.e. the total amount of travelling between places is minimized
- a_2 - if not explicitly stated, a user prefers the itineraries that contain more places, about which the user expressed positive opinion in her preferences
- a_3 - the preferences of the user are expressed in the interval scale
- a_4 - the time gap between two dinings that is smaller than 4 hours is unacceptable for a user

While the a_1 assumption is straightforward, on the other hand the a_2 and the a_3 assumptions need an elaboration on a more detailed level. Since the heuristic approach for the OPMTW problem proposed in the section 4.2 works on the graph where each vertex has its profit value per user, we have to map the user preferences to the profit values of the vertices on the graph. We do this by collecting the explicit feedback from the users by letting them choose the benefit value for each possible category of the objects in a graph. In our case the benefit value for each category is expressed by a number from 1 to 100, therefore each vertex on a graph will get assigned its profit as a number equal to its category benefit value.

As stated before, when constructing the itinerary, the algorithm will try to maximize the profit of all visited vertices. Given the fact, that we allow the user to choose the preferences in the interval scale, there might be the instances of the constructed itineraries where the priority is given to the bigger amount of vertices with lower profit values than to the smaller amount of vertices with higher profit values. Consider an example of two insertion alternatives. In the first case two nodes v_a and v_b are inserted into the path, where both vertices v_a and v_b belong to the category with the benefit value of 75 (thus $v_a = v_b = 75$). In the second case the node v_c is inserted into the path, where the vertex v_c belongs to the category with the benefit value of 100 ($v_c = 100$). Based on our interval scale assumption for the preferences, the insertion of the vertices v_a and v_b will be preferred over the insertion of a vertex v_c, because the overall profit of v_a and v_b would be 150, while the profit of v_c would be 100

The validity of the assumption a_3 is argueable and its substitution with the one that uses the ordinal scale for the preferences can be considered.

4.4.1 Intensity of the Trip

Additional user preference that is taken into account when designing the application is the intensity of the trip. The heuristic approach that was described earlier has the scope of maximizing the profit of the constructed itinerary, thus it will try to fit as many and as profitable objects within the travel time span specified by the user. However the tourists often are not willing to rush through a city visiting as many places as possible, even if they are interested in the potential objects and they can afford the travel of the itinerary time-wise. In some cases the tourists might enjoy the itineraries that offer some free time gaps inbetween the visits of the objects. For this motivation we allow the users of the system to specify their preferred intensity of the trip by selecting a numeric value. Using heuristics this value is mapped to the maximal number of the POIs that can be suggested to the user per path and used as a parameter in the described heuristic approach to solve the Orienteering Problem with Multiple Time Windows.

4.4.2 Dining Constraint

Since all eating places in our application is modelled in the same way as all other POIs, the algorithm might suggest many of them, given the high user preferences for dining places and the satisfiability of the time constraints. However we do not want a user to eat 4 or 5 times per day, no matter what are the preferences for the specific types of restaurants. For this reason in order to better represent the real life scenarios we introduce the so called dining constraint. The dining constraint makes the insertion of a POI that belongs to the dining category infeasible, if there is another dining POI already inserted in the path P within 4 hours time distance from the arrival time to the first POI.

4.5 Generating k-unrepetitive Itineraries

As discussed before, some assumptions had to be made about the users before designing the system. The assumptions included the statements that a user prefers the shorter itineraries as well as the itineraries that have higher overall profit value. However the assumption of the overall profit value had always had the precedence over the shorter-trip assumption. To better represent the user needs about the priorities of the assumptions, the user is given the freedom of choosing between several itineraries that differ both in the profit and in the cost of the trip and in this way the user can decide himself which itinerary provides more satisfaction.

One solution to address the similar problem was suggested by Lederer et. al [21]. His approach was to generate many routs by introducing penalty for the route elements that are already present in another route. In this way, Lederer's modificaiton of A* algorithm was able to produce not only the optimal route, but also other alternatives that might be satisfactory for the user.

In our case the problem is addressed by generating up to k unrepetitive paths P in the developed application, where k is the arbitrary parameter. The generation procedure of k paths is performed using the earlier mentioned heuristic approach for the OPMTW and the different values of the preference parameters. After each iteration i, the benefit values are decreased for every POI, that was inserted in the path P_i. The procedure is repeated t times, where t is the arbitrary parameter. Then out of t generated paths, k_1 unrepetitive paths are selected based on the overall profit and k_2 unrepetitive paths are selected based on the travel cost, where $k_1 + k_2 = k$.

After the generation, k-unrepetitive itineraries are suggested to the user. Some of them will not be optimal based on the time cost or the overall profit of the visited POIs, however they might be still more satisfactory to the user than the optimal one.

5 Implementation

This section presents the design and development process of the itinerary planning system aimed for the visitors and tourists of South Tyrol. The system consists of the two main software components: the heuristic algorithm running on a server, and a mobile application that serves as a client for the preference elicitation from the user and the visual display of the results. The development process started with the definition of the system requirements, the main functionalities and the analysis of possible development approaches. After the completion of these steps the appropriate system architecture that fulfills all the stated requirements as well as implements all the defined functionality has been designed. In this section both components of the system are described.

5.1 Server-side

Package Name	**Description**
opmtw.web	Servlets and supporting classes
opmtw.db	Database connection
opmtw.pathconstruction	The main classes for the computation of the algorithm
opmtw.datamodel	The object classes
opmtw.geo	Distance calculation between objects
opmtw.json	JSON parsing from the Google API responses

Table 1: The packages of the source code for the JAVA application on the server side

The server side has been implemented using Java programming language and Eclipse IDE. The SQLite 3.7.6 database was used for storing the information about the POIs and the cost information. Apache webserver has been used together with the Tomcat 7.0.6 servlet container. Two servlets have been developed in order to respond to the HTTP requests: POIServlet and ItineraryServlet. The main responsible of the POIServlet is to send the list of available start and end points to the user. The responsability of ItineraryServlet was to send the top-k unrepetitive itineraries to the client. All responses were send as an HTTP resonse in JSON format. The package structure of the main application is shown in Table 1.

5.2 Mobile Client for iPhone

The mobile application *SmartTrip* for the iOS devices has been developed. While it was targeted for the iPhone users, it can be used on any device running the iOS 4.2 or higher operating system, what currently includes the different variety of iPhones, iPod Touches and iPads.

5.2.1 Technologies Used

The mobile application was developed in Objective-C programming language using Cocoa Touch Apple's proprietary objective-oriented API. As the development environment the XCode 4.2 was used together with the iOS 4.3 SDK. The application was developed following the iOS Human Interface Guidelines [24] that describe the principles of the

Package Name	Description
itinerary.model	Stores the model classes of the project representing objects, such as a POI or an itinerary
itinerary.json	JSON parsing and constructing model objects
itinerary.network	Connection management
itinerary.viewcontrollers	Manipulation of the graphical interface
itinerary.logic	The main calculation classes of the mobile app

Table 2: The packages of the source code for the mobile application

superlative user interface and user experience design for the iOS applications. The user interface for the mobile application has been designed using the Interface Builder - an integrated tool within the XCode 4.2 development environment.

The communication with the server is implemented over the HTTP protocol. The request for the retrieval of the starting/ending points and the request for the itineraries generation are send using the HTTP GET request method. The response from server arrives in JSON format.

The *SmartTrip* application is divided into 5 packages located in the source folder of the project (See Table 2).

5.2.2 Mobile Application Usage Scenario

A typical application usage scenario of the *SmartTrip* proceeds as follows. On the launch the application tries to connect to the server and download the list of the places in the city from which a user can start the trip. The same list is used for the selection of the end of the trip. After the launch of the application the user is immediately presented with the home-settings screen (See the leftmost image in Figure 16), where she can choose the compulsory preferences (such as the start point and the end point of the trip, the travel date, the starting and the ending times) and the optional ones (the preferences of the places to visit, the eating preferences and the intensity of the trip). In case if user ignores the optional preferences, the request for the itinerary can still be send. In this case the default preference values are used.

If the user decides to express her preferences before sending the request for the itinerary computation, she can do so by specifying the nominal values for different categories, specifying her interest in them (See the middle image in Figure 16). After the user clicks the "Calculate" button, the HTTP Get request is sent over to the server with the specified trip parameters. When the generation of the itineraries is finished on the server side, the mobile application receives the response in JSON format. An example of the response is displayed in the figure (See Figure 18).

The first out of maximum 3 itineraries is immediately displayed in a map view for the user. Each suggested POI contains additional information (such as its name, visiting order, arrival and departure times) which can be view by clicking on the points in the map. All points are connected by a virtual path based on the visiting order of the POIs. The lines in the path do not represent the actual walking directions on the street level but rather indicate the sequence of the points. More detailed itinerary information for every trip can be viewed in the detail view (See the middle image in Figure 17). The user can explore the order of the POIs to visit, their name, description and arrival/departure

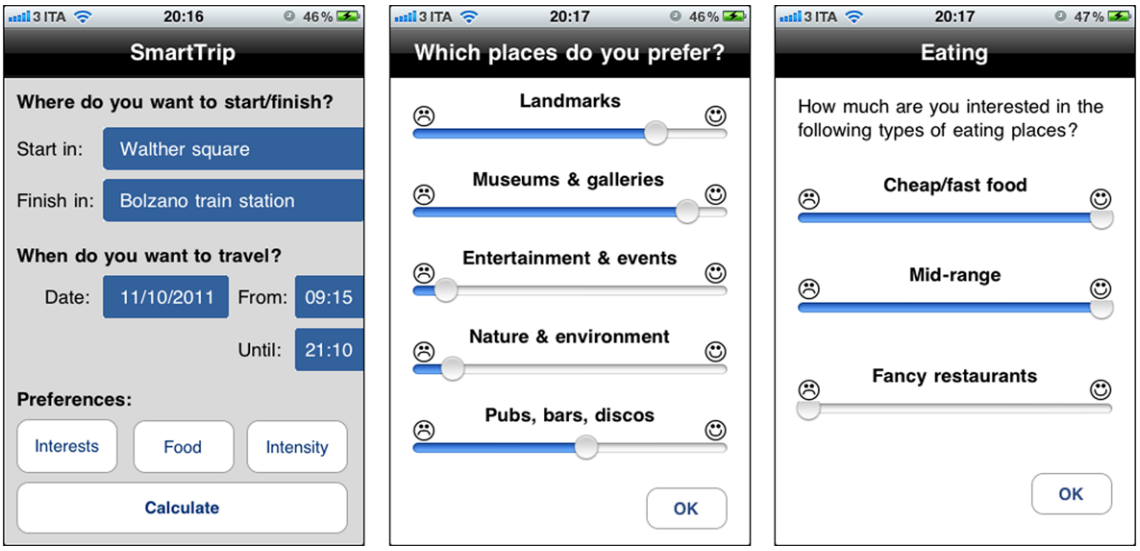


Figure 16: The screenshots of the *SmartTrip* app: on the left - the home screen, in the middle - POI preferences, on the right - restaurant preferences

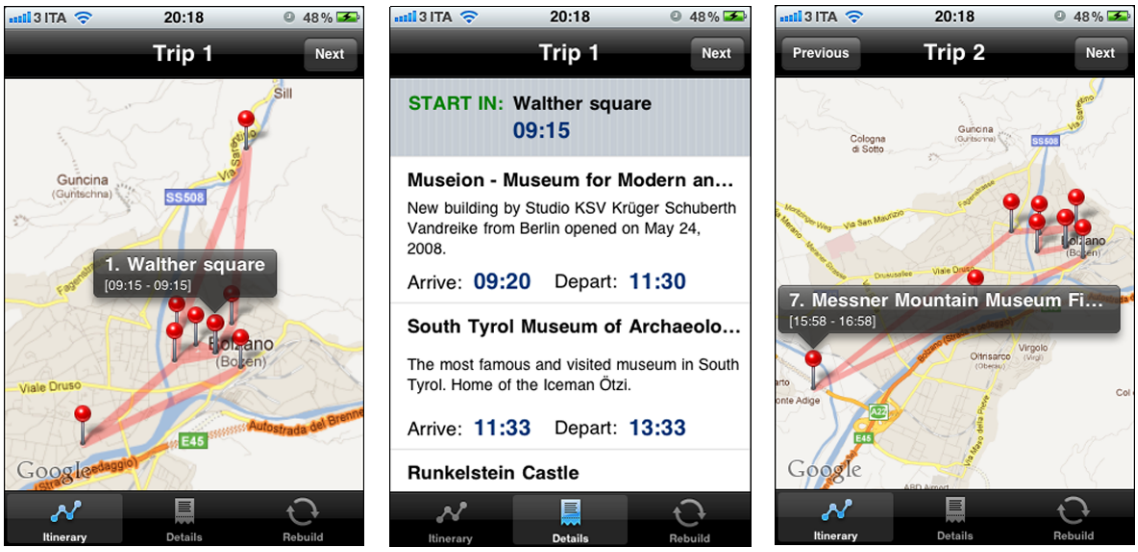


Figure 17: The screenshots of the *SmartTrip* app: recommended itineraries showed on the map and in the detail view

times. Any time before or during the trip a user can rebuild the itineraries by tweaking one or more parameters.

```
[{"pois":[{"arrivalTime":540,"departureTime":540,"poiID":17},{"arrivalTime":541,"departureTime":690,"poiID":2},
{"arrivalTime":693,"departureTime":960,"poiID":12},{"arrivalTime":963,"departureTime":993,"poiID":7},
{"arrivalTime":1002,"departureTime":1070,"poiID":8},{"arrivalTime":1074,"departureTime":1089,"poiID":4},
{"arrivalTime":1091,"poiID":18}],"cost":22},
{"pois":[{"arrivalTime":540,"departureTime":540,"poiID":17},{"arrivalTime":541,"departureTime":690,"poiID":2},
{"arrivalTime":699,"departureTime":810,"poiID":9},{"arrivalTime":822,"departureTime":852,"poiID":7},
{"arrivalTime":854,"departureTime":960,"poiID":12},{"arrivalTime":966,"departureTime":981,"poiID":4},
{"arrivalTime":983,"poiID":18}],"cost":32}
{"pois":[{"arrivalTime":540,"departureTime":540,"poiID":17},{"arrivalTime":541,"departureTime":690,"poiID":2},
{"arrivalTime":699,"departureTime":810,"poiID":9},{"arrivalTime":828,"departureTime":888,"poiID":13},
{"arrivalTime":896,"departureTime":960,"poiID":12},{"arrivalTime":963,"departureTime":1080,"poiID":10},
{"arrivalTime":1086,"poiID":18}],"cost":45}]
```


Figure 18: A server JSON response containing the information about the generated itineraries

6 Evaluation

In this chapter we present both the quantitative evaluation performed on the heuristic approach to solve the Orienteering Problem with Multiple Time Windows and the qualitative approach to evaluate the developed system and the assumptions considered beforehand.

6.1 Quantitative Evaluation

6.1.1 Data

The heuristic approach for the Orienteering Problem with Multiple Time Windows has been tested using the well-known Solomon's data-set of VRPTW instances [22]. Even if the dataset has been initially designed to test solutions to the Vehicle Routing Problem with Time Windows (VRPTW), as stated by Solomon the design of the dataset highlights several factors that affect the behavior also of the other routing and scheduling algorithms. Three sets of problems R1-type, C-1 type and RC-1 type have been used for the testing purposes. Each dataset has multiple problems, where each problems consists of the following data: the customer id, x coordinate, y coordinate, the demand, the ready time, the due date and the service time. This data can be easily mapped to our data model, where x and y coordinates correspond to the latitude and the longitude of the POIs, the demand corresponds to the profit visiting a POI, the ready time and the due data represent the single time window per POI and the service time corresponds to the visiting time of a POI.

The difference between three data sets is the generation of the geographical data. In the problem set R1 the geographic coordinates were generated randomly, in the set C-1 the instances of customers were clustered geographically, while in the set RC-1 there is a mix of random and clustered structures. The coordinates of customers are identical for all problems within one data set type. Some have very tight time windows, while others have time windows which are hardly constraining. The problems are 100 customer euclidean problems where travel times equal the corresponding distances.

It should be noted that even if the data was designed to be tested on algorithms that address problems with single time windows, the data are perfectly fine for the test with multiple time windows, as it is done in our case.

6.1.2 Experimental Results

The tables 3, 4 and 5 compare the scores obtained by the OPMTW heuristics with both the previously presented ILS heuristic and the optimal solutions for the TOPTW instances. Column one is the name of the problem instance. Column two and three represent the profit and the amount of visited vertices for the TOPTW optimal solutions. We are taking only the results with m=1, where m is amount of vehicles/agents used in TOPTW, thus generalizing the TOPTW to the OPTW. Columns 4-7 report data for the ILS heuristics, namely the profit, the amount of vertices, the gap between the optimal

Instance	Optimum		ILS				OPMTW			
Name	Profit	Vertices	Profit	Vertices	Gap (%)	CPU (s)	Profit	Vertices	Gap (%)	CPU (s)
c101	320	10	320	10	0.0	0.4	290	12	9.4	0.42
c102	360	11	360	11	0.0	0.3	240	12	33.3	**0.27**
c103	400	11	390	10	2.5	0.5	330	11	17.5	**0.3**
c104	420	11	400	10	4.8	0.3	350	11	16.6	0.32
c105	340	10	340	10	0.0	0.3	310	12	8.8	**0.25**
c106	340	10	340	10	0.0	0.3	300	11	11.8	**0.23**
c107	370	11	360	11	2.7	0.3	310	11	16.2	**0.25**
c108	370	11	370	11	0.0	0.3	360	12	2.7	0.31
c109	380	11	380	11	0.0	0.3	340	11	10.5	**0.27**

Table 3: Results of Solomon's test problems with random geographic data, (m = 1)

Instance	Optimum		ILS				OPMTW			
Name	Profit	Vertices	Profit	Vertices	Gap (%)	CPU (s)	Profit	Vertices	Gap (%)	CPU (s)
r101	198	9	182	7	8.1	0.1	153	7	22.3	0.11
r102	286	11	286	11	0.0	0.2	275	11	3.8	0.3
r103	293	11	286	10	2.4	0.2	266	10	9.2	0.29
r104	303	12	297	11	2.0	0.2	220	9	27.4	0.21
r105	247	11	247	11	0.0	0.1	212	9	14.2	0.21
r106	293	11	293	11	0.0	0.2	177	7	39.6	**0.14**
r107	299	13	288	10	3.7	0.2	257	10	14	0.28
r108	308	12	297	11	3.6	0.2	224	9	27.3	0.22
r109	277	12	276	11	0.4	0.2	204	8	26.4	0.2
r110	284	13	281	11	1.1	0.3	276	11	2.8	0.3
r111	297	12	295	11	0.7	0.2	216	8	27.3	0.23
r112	298	12	295	11	1.0	0.2	277	11	7	0.32

Table 4: Results of Solomon's test problems with clustered geographic data, (m = 1)

solution and the ILS solution in percentage and the CPU time taken to solve the problem. The last 4 columns report the same data about our OPMTW heuristic.

All computations for the ILS heuristics were carried out on a personal computer with Intel Core 2 2.5 GHz processor and 3.45 GB Ram. All computations for the OPMTW heuristics were carried out on a personal computer with Intel Core i5-460M 2.53 GHz processor and 4.00 GB of Ram. The computational power that was used for both set of computations is very similar thus we can objectively compare not only the correctness of the computations but the computational times as well.

The average gap between the result of the ILS heuristics and the optimal solution is only 1.9% for all the problem instances. In the worst case the gap is 9.4%. On the hand, the average gap between the optimal solution and the OPMTW heuristics is 15.8% with the worst case reaching 39.6%. Even if the ILS outperforms the OPMTW heuristic approach, the later still provides high quality results that are quite close to the optimal ones.

The average computation time for all problem instances is the same for both ILS and OPMTW approaches (0.24s). However the OPMTW approach outperforms the ILS heuristic in two out of three problem set types. For the C1 type of problems the av-

Instance	Optimum		ILS				OPMTW			
Name	Profit	Vertices	Profit	Vertices	Gap (%)	CPU (s)	Profit	Vertices	Gap (%)	CPU (s)
rc101	219	9	219	9	0.0	0.2	214	10	2.3	0.21
rc102	266	10	259	9	2.6	0.2	247	11	7.1	0.23
rc103	266	10	265	11	0.4	0.3	226	9	15	**0.2**
rc104	301	11	297	11	1.3	0.3	255	10	15.3	**0.26**
rc105	244	11	221	11	9.4	0.2	231	10	5.4	0.23
rc106	252	11	239	11	5.2	0.2	188	7	25.4	**0.14**
rc107	277	10	274	11	1.1	0.2	206	7	25.6	**0.16**
rc108	298	11	288	11	3.4	0.2	222	8	25.5	**0.19**

Table 5: Results of Solomon's test problems with a mix of random and clustered geographic data, (m = 1)

erage computation time for ILS is 0.33s, OPMTW 2.62s. For the type R1: ILS 0.19s, OPMTW 0.23s. For the type RC1: ILS 0.23s, OPMTW 0.2s. OPMTW outperforms ILS in 12 out of 29 problems (in the tables marked as bold numbers), and performs worse than ILS only for the R type of problems.

In conclusion the ILS approach returns higher quality results than OPMTW, however the results of the OPMTW are still good and close to the optimal results. The computation time of the OPMTW is faster for the data sets where object instances are clustered geographically or clustered objects are mixed with the random objects. For the objets with randomly generated geographic coordinates the ILS approach finds the results faster.

6.2 Qualitative Evaluation

6.2.1 Used Methodology

We chose to evaluate the solution qualitatively in order to understand how good the solution is for the user. However before the development of the system some assumptions had to be made what does "good" mean for the user. The assumptions have been based on the literature and author's personal knowledge.

In order to verify the stated assumptions first of all the appropriate empirical method for the investigation had to be chosen. Based on the need for our system the indirect expert based evaluation has been chosen as the empirical method to evaluate the system qualitatively. Between several reasons of the decision to choose the expert based evaluation some include: the availability of the experts and higher validity of the evaluation. The validity can be guaranteed because an expert can cover the knowledge about various types of the users for different age groups, nationalities and interests, whereas performing a direct experiment with a selected sample from a population could result in biased and imprecise conclusions.

The system presented in this paper was evaluated qualitatively by the tourism expert Nadja Sternkopf, who is the member of the Info Team in the main tourist office of Bolzano. The evaluation was conducted in a format of the interview. The expert used the system and estimated the satisfaction of the potential users, given their preferences and constraints. Moreover the expert evaluated the assumptions that have been made in order to design the application. The results of the evaluation are summarized in the following

subsection.

6.2.2 Outcome

Based on the statistical data kindly provided by the Central Tourist Office of Bolzano (Azzienda di Soggiorno e Turismo, Piazza Walther-Platz 8), the medium duration of staying at the hotels of Bolzano was only 1,7% days in the first 6 months of 2011 and 1,8% in the first 6 months of 2010. The statistics suggest that tourists usually stay in the city just for a very short time period, and thus it can be derived that they are more interested in day routes rather than the recommendations for a week. This was also confirmed by the expert working in the tourism office, who also underlined that the question that is asked by the majority of the tourists is the itinerary suggestion for one day within the city. This can be explained for the specific case of Bolzano city by the fact, that most of travellers prefer to stay a longer time in the mountains surrounding Bolzano, and leave just a short time for visiting the city itself.

When designing the system several assumptions have been taken into account, two of which are the following:

- a_1 - a user prefers the itineraries that are shorter in distance, i.e. the total amount of travelling between places is minimized
- a_2 - if not explicitly stated, a user prefers the itineraries that contain more places, about which the user expressed positive opinion in her preferences

However the relationship was not specified between the two in case of the contradicting scenarios, for example how much the user is willing to sacrifice the a_1 assumption for the a_2 assumption, i.e. how much would a user walk in order to reach his preferred place of interest, given its relatively long distance from the actual user position. As stated by the expert, users are more often willing to sacrifice for the a_2 assumption, i.e. they prefer to skip the places that are of great interest to them if the distance/cost to reach them is not reasonable.

The potential pitfalls were considered for both cases: when a tourist asks for the route in the city to an expert (in this case, a person working in the tourism office) and when a tourist relies on our system for the itinerary planning and does not interact with the epxerts. Several aspects were considered when comparing the two mentioned recommendation sources. First of all, when suggesting the itineraries the experts more often take into account the given time constraints by the users rather than their preferences. For example, if a tourist has only 2-3 hours in the city, the expert has predefined routes to suggest for him given his time constraints, independently from the possible wishes of the tourist. In this case the preferences of the tourist are ignored, and any potential alternative path is not considered, which would possibly still satisfy the given time constraints and be more satisfactory for the user.

Another disadvantage of consulting an expert rather than the itinerary planning system are the legal issues. Even if an expert has the sufficient knowledge of the restaurants and is given the exact preferences of a tourist, the expert cannot advice a specific place for eating because of the legal terms imposed by the law. The expert, being a part of the tourist association, has to remain neutral when giving advices about businesses, such as restaurants. To assist the tourist with the restaurant suggestions the experts usually give the set of places within the specific geographic region (e.g. an old tow, or a city

center), provides the neutral information about all of them and finally lets the user to make the final decision. On the other hand, the system is not constrained by such legal terms and therefore can provide a better recommendation by taking into account the user preferences.

6.2.3 Comparison Analysis

To evaluate the effectiveness of the itinerary planning by the system the following experiment has been conducted. The expert was asked to define a request, that is typically expressed by a tourist, stating the typical preferences and constraints. The expert then was asked to use the system on behalf of the tourist and estimate the satisfaction of the potential user.

The following preferences and constraints identifying the typical tourist request have been stated by the expert:

Start time - 9am. End time - 6pm. Start point - Walther square. End point - Walther square.

Preferences:

Landmarks - 100/100. Musuems&galleries - 100/100. Entertainment&events - 50/100. Nature&environment - 50/100. Pubs, bars, discos - 0/100. Cheap restaurants - 65/100. Mid-range restaurants - 50/100. Fancy restaurants - 0/100.

The three itineraries that have been generated by the system are displayed in the figure 19.

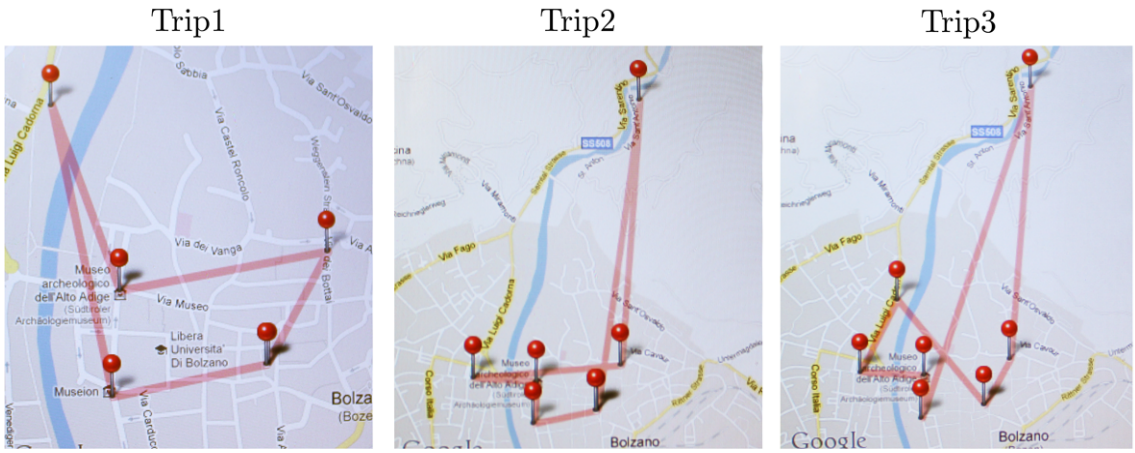


Figure 19: The three itineraries generated by the system based on the preferences specified by the expert

The general feedback expressed by the expert toward all three generated itineraries was positive, as all of them would be feasible for a potential typical tourist. However the feedback included multiple notes. First of all despite the high preference value for the category of "Museums&Galleries" the expert expressed the potential negative opinion of a user about the adding of the Museum of Modern Art for all three itineraries. The main motivation was the general unpopularity of the museum itself, even between the tourist groups who enjoy museums. This implies that in order to improve on this aspect

the system can be advanced in one of the two ways: either the subcategories have to be considered for some of the categories (e.g. the museums have to be split into the subcategories that define the museum's type) or additional parameter which defines the general popularity of a place should be introduced. The latter approach could be more feasible if the general popularity score would be used in combination with the user's preference toward the category of that place.

Out of all three itineraries the expert chose the second one as the best for the potential tourist. The main reasons why the second option could be more satisfactory for the tourist were the following: the exploration of the different city disctrict (by going out of the old town to visit the Victory Monument) and the visit of the castel Roncolo, which is considered to be one of the top attractions in the city and can be reached by the beautiful path that leads to it. The second option was preferred over the third one mainly because of the suggested visiting time for the recommended restaurant - in the third option the time to visit it was chosen in the late afternoon, which can be unacceptable for some tourists, as stated by the expert.

One of the suggestions by the expert with respect to the usability of the system was missing consideration of the number of times a tourist has already been to a given city, because the recommended itinerary can depend on this factor too.

Taking into account the feedback provided by the expert, the following conclusions have been drawn about the itinerary planning approach. By analizing the different itineraries the expert implicitly suggested that the paths connecting the points of interest can be as much important as the points of interest themselves. For example, the tourists can be satisfied with the long distances that lead to the points of interest, as long as the path itself gives satisfaction to the user. Additional benefit can be added for exploring new areas, such as the new city districts. Therefore in order to improve the system in the future, it would be possible to assign not only the cost for the edges that connect the points of interest but also the benefits, which could be determined dynamically in the same way as the benefit for points of interest.

7 Conclusions and Future Work

Planning a good itinerary in an unknown city is a difficult task for a tourist to solve by herself. There are usually a plenty of factors to consider when trying to pick a satisfactory places to visit, such as their types, the reachability, the opening hours etc. Moroever the task becomes even more challenging when an itinerary has to be chosen which connects all of the selected points in such a way that a tourist would not need to travel excessively and the trip would be feasible with respect to the opening hours of the places. Normally the LTOs (Local Tourist Organizations) try to help the tourists to address the mentioned task. However LTOs usually tend to partially ignore the user profile due to the time limitations, and they are restricted by the legal constraints, which prevent them from suggesting the specific businesses (such as restaurants) explicitly in order to maintain the LTO's neutrality.

In order to better address the problem of the itinerary planning for tourists when the opening times of the places are important a new routing problem was introduced in this work. The Orienteering Problem with Multiple Time Windows (OPMTW) combines the objectives and constraints of the existing routing and scheduling problems such as Orienteering Problem, Vehicle Routing Problem and Travelling Salesman Problem with Time Windows. An effective and fast iterative local search heuristic method was developed in order to solve the OPMTW. The heuristic was evaluated quantitavely using the well-known Solomon's data-set of the instances for the Vehicle Routing Problem with Time Windows. The results of the OPMTW were compared both to the results of the ILS heuristic and the optimal solutions. The performed evaluation showed that the average profit gap between the presented OPMTW approach and the optimal solutions is 15,8%, with the worst case reaching only 39.6%. Moreover the OPMTW approach outperformed the ILS heuristics in two out of three problem set types, showing the superior performance of the OPMTW for the datasets where the objects were clustered geographically.

The heuristic algorithm was applied in the development of the itinerary planning system for recommending multiple unrepetitive trips, based on the user preferences, time constraints and the context-based information. The algorithm has been implemented for the client-server architecture together with an application for mobile devices with the iOS operating system, that is able to generate multiple itineraries in a feasible time for a user. The developed system has been qualitatively evaluated by an expert working in a tourism field.

Based on the output of the evaluation, the following potential future improvements have been considered. First of all in order to better assess the user needs the formulation of the problem can be modified by adding the profit weights not only to the nodes in a graph, but also to the edges connecting them. Another potential improvement that could be considered in the future is the adding of the common-profit parameter for every node in the graph, which would be static for all tourists independently of their preferences. Morover the number of times a tourist has alread been to a given city could be considered, because the suggestion of an itinerary can also depend on this factor.

References

[1] Vansteenwegen, P., Van Oudheusden, D. (2007). The mobile tourist guide: an OR opportunity. OR Insights; 20(3):217.

[2] Mittenthal, J. and Noon, C.E., (1992). 'An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint'. Journal of the Operational Research Society, 43(3): 277-283.

[3] Baldacci, R., Hadjiconstantinou, E., and Mingozzi, A. An exact algorithm for the Traveling Salesman Problem with Deliveries and Collections. In Proceedings of Networks. 2003, 26-41.

[4] Lawrence V. Snyder, Mark S. Daskin, A random-key genetic algorithm for the generalized traveling salesman problem, European Journal of Operational Research, Volume 174, Issue 1, 1 October 2006, Pages 38-53, ISSN 0377-2217, 10.1016/j.ejor.2004.09.057.

[5] Jozefowiez, N., Glover, F., and Laguna, M. (2008). Multi-objective Meta-heuristics for the Traveling Salesman Problem with Profits. Journal of Mathematical Modelling and Algorithms 7, 177-195.

[6] Baraglia, R.; Hidalgo, J.I.; Perego, R.; , 'A hybrid heuristic for the traveling salesman problem', Evolutionary Computation, IEEE Transactions on , vol.5, no.6, pp.613-622.

[7] E. Balas. The Prize Collecting Traveling Salesman Problem. Networks, 19:621–636, 1989.

[8] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In Proceedings of the 27th Annual ACM Symposium on Theory of Computing, pages 277-283, May 1995.

[9] Matthias Joest and Wolfgang Stille. 2002. A user-aware tour proposal framework using a hybrid optimization approach. In Proceedings of the 10th ACM international symposium on Advances in geographic information systems (GIS '02). ACM, New York, NY, USA, 81-87.

[10] T. Tsiligrides, 'Heuristic methods applied to orienteering', Journal of the Operational Re-search Society 35 (1984), 797-809.

[11] Gilbert Laporte, The vehicle routing problem: An overview of exact and approximate algorithms, European Journal of Operational Research, Volume 59, Issue 3, 25 June 1992, Pages 345-358.

[12] Lpez-Ibez, M., Blum, C., Beam-ACO for the travelling salesman problem with time windows, Computers & Operations Research, 37 (9) (2010) 1570-1583.

[13] Savelsberg, M.W.P. 1985. Local search in routing problems with time windows. Annals of Operations Research 4 285-305.

[14] Focacci, F., Lodi, A., Milano, M. A hybrid exact algorithm for the TSPTW. INFORMS Journal on Computing, 14 4 (2002), pp. 403417.

[15] Righini, G., Salani, M. Decremental state space relaxation strategies and initialization heuristics for solving the Orienteering Problem with Time Windows with dynamic programming, Computers & Operations Research, Volume 36, Issue 4, April 2009, Pages 1191-1203.

[16] Vansteenwegen, P., Souffriau, W., Berghe, G. W.,Van Oudheusden, D. Iterated local search for the team orienteering problem with time windows, Computers & Operations Research, Volume 36, Issue 12, December 2009, Pages 3281-3290.

[17] Silver, E.A., (2002) 'An overview of heuristics solution methods', The 7th Annual International Conference on Industrial Engineering, Busan, Korea

[18] B. L. Golden, L. Levy and R. Vohra (1987) The orienteering problem. Naval Res. Logist. 34, 307-318.

[19] M. Gendreau, A. Hertz, G. Laporte and M. Stan (1992) New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. Operations Research Vol. 40, No. 6 (Nov. - Dec., 1992), pp. 1086-1094

[20] M. Gendreau, A. Hertz, G. Laporte and M. Stan (1998) A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. Operations Research Vol. 46, No. 3 (May - Jun., 1998), pp. 330-335

[21] Lederer, T. Development of Navigation Systems. (2009) Saarbrcken: VDM Verlag.

[22] Solomon M. (1987) Algorithms for the vehicle routing and scheduling problem with time window constraints. OperationsResearch 1987; 35:25465.

[23] C. Robson. Real World Research: A Resource for Social Scientists and Practitioners-Researchers, Blackwell (1993).

[24] iOS Human Interface Guidelines, (2011), Apple Inc. http://developer.apple.com/library/ios/documentation/userexperience/mobilehig

[25] Flickr API, (2011), http://www.flickr.com/services/api/