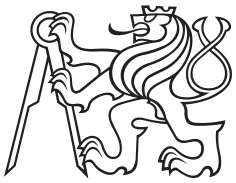
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Automated planning of tourist trips with a desired length

Bc. Jan Pancíř

Supervisor: Doc. Ing. Michal Jakob, Ph.D.
Field of study: Open informatics
Subfield: Artificial intelligence
May 2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Pancíř Jan** Personal ID number: **457925**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master's thesis details

Master's thesis title in English:

Automated planning of tourist trips with a desired length

Master's thesis title in Czech:

Automatické plánování výletů požadované délky

Guidelines:

Automated trip planning is an algorithmically challenging and application-relevant open problem. The goal of planning tourist trips is to generate routes which pass through attractive locations and route segments while respecting user preferences on the ideal length of the trip. When solving the problem, please proceed as follows:
1. Familiarize yourself with existing approaches to tourist trip planning
2. Formalize the tourist trip planning as an optimization problem.
3. Design and implement an algorithm for solving the tourist trip planning problem.
4. Use the algorithm to build a prototype of a working tourist trip planning system.
5. Evaluate the performance of the developed system on real-world data and discuss the results.

Bibliography / sources:

[1] Gionis, Aristides, et al. "Customized tour recommendations in urban areas." Proceedings of the 7th ACM international conference on Web search and data mining. ACM, 2014.
[2] Gavalas, Damianos, et al. "Heuristics for the time dependent team orienteering problem: Application to tourist route planning." Computers & Operations Research 62 (2015): 36-50.
Souffriau, Wouter, and Pieter Vansteenwegen. "Tourist trip planning functionalities: State–of–the–art and future." International Conference on Web Engineering. Springer, Berlin, Heidelberg, 2010.

Name and workplace of master's thesis supervisor:

doc. Ing. Michal Jakob, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **04.02.2020** Deadline for master's thesis submission: _____

Assignment valid until: **30.09.2021**

doc. Ing. Michal Jakob, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to thank doc. Ing. Michal Jakob, Ph.D. for supervising my thesis and providing me with helpful ideas and valuable comments on this topic which is very close to me. I would also like to thank my family and my closest friends with whom I have experienced exciting tours to many incredibly beautiful places.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Praha/Prague, 20. May 2020

Abstract

In this thesis, approaches for the automated tourist trip planning with a desired length are studied, especially solutions for the orienteering problem (OP) and its extensions. I define my own planning task resulting in tours on maps which visit the most interesting places and lead along the most interesting paths. I design an algorithm solving this task based on a combination of the evolution approach and the metaheuristic Variable Neighborhood Search algorithm (VNS). I design the Automated Trip Planning System (ATPS) based on this algorithm. I also design an optimal algorithm based on linear programming. I test results on a custom test benchmark set and also on a real-map data from several locations in the Czech Republic. The achieved results proof the usability and effectivity of many tour recommendations within a few seconds.

Keywords: automated trip planner, orienteering problem, multi-trip planning, fast trip planning

Supervisor: Doc. Ing. Michal Jakob, Ph.D.

Abstrakt

V této práci jsou studovány přístupy k automatizovanému plánování turistických výletů s požadovanou délkou, zejména řešení problémů orientace (OP) a jejich rozšířeními. Definuji vlastní plánovací úlohu jejíž výsledkem jsou trasy v mapě, které navštěvují ta nejzajímavější místa a vedou po těch nejzajímavějších cestách. Navrhuji algoritmus řešící tuto úlohu založený na kombinaci evolučního přístupu a metaheuristického algoritmu Variable Neighborhood Search (VNS). Navrhuji automatizovaný systém pro plánování turistických výletů (ATPS) založený na tomto algoritmu. Navrhuji také optimální algoritmus založený na lineárním programování. Výsledky testuji na umělé testovací sadě a poté na reálných mapových podkladech z několika lokalit z České Republiky. Dosažené výsledky dokazují použitelnost a efektivitu systému v doporučování mnoha turistických tras během několika sekund.

Klíčová slova: automatizovaný plánovač výletů, problematika orientace, plánování více tras, rychlé plánování tras

Překlad názvu: Automatizované plánování turistických výletů požadované délky

Contents

Figures

Tables

Chapter 1

Introduction

The automated trip planning is an algorithmically challenging and application-relevant open problem. Every day, many people in the world plan their vacations and leisure trips to many diverse places on the Earth. But looking at the map and inventing the best route might not be an easy task, especially in a location unknown to the tourist. With a huge amount of data gathered so far and well-described maps which we have nowadays, the automated trip planning can not only save a lot of time for many tourists, but it can also create trips which can compete or possibly surpass trips created by hand of an experienced human trip creator.

The main goal of the trip planning is to find pleasant routes. It is an objective that is hard to model and very hard to measure precisely. This objective description is the main difference from the majority of other optimization tasks where usually the goal is to minimize a well-described objective function. In case of the trip planning, designing of an objective function is not so obvious and it can deviate significantly from the actual purpose of the planning task in real-life problems. The proper way to say that one route is better than the other is the key to success for a good trip planning solver.

Solving the trip planning task consists of two main steps. First we need to find a set of interesting locations to visit. These locations might correspond to viewpoints, alpine huts, cultural sites and many more different attractions. However, we also want to include some interesting paths, e.g. trail through a rocky canyon is definitely more interesting than a path along a straight highway. A good selection of these locations and segments on the map is an important starting point for the next step. In the second step, we need to

find a path between the user-defined start and goal locations, not longer than the user-provided time budget, and visiting as many interesting locations and segments as possible at the same time. Since usually not all locations and segments can be visited at once due to the time budget, not only the path is needed, but also the determination of a subset of these places has to be made, and thus this problem belongs to the class of NP-hard problems.

In this thesis, I design and propose a system for the automated trip planning. Firstly I focus on the analysis of possible ways to achieve the automated trip planning. I provide an overview of several solutions and concepts used in this area. Then I define my trip planning task with points and segments of interest the solver is created for. In the chapter 4, I design the Tour Searching (TS) heuristic algorithm and the Automated Trip Planning System called ATPS which is based on this algorithm. The system provides fast top K distinct trip recommendations based on user requirements. In addition, it enables the user to adjust the resulting trips by simple location preferences updates. In the chapter 5, I design an Integer Linear Programming (ILP) algorithm which finds an optimal solution for the defined task. In the chapter 6, I describe data gathering and implementation approaches. In the chapter 7, I describe experiments and testing performed with this system using the heuristic and the optimal algorithm, and discuss the achievements which prove the system to be usable for several real-life trip planning problems.

Chapter 2

Related work

A huge amount of research has been done in tasks related to trip planning or trip recommendation. The proposed approaches are usually specialized for specific real-life problems and differ in functionality. An overview of current functionalities was well mapped by Wouter Souffriau and Pieter Vansteenwegen [1]. In this chapter, I analyze and describe different existing approaches and solutions for the trip planning tasks, especially approaches related to the automated tours generation. I use some concepts of these approaches for my automated trip planning system designed in the following chapters.

2.1 Trip planners

Several trip planning solvers have been developed to solve some variations or specifications of a general trip planning task. These solutions variate from the data used, the method used and the actual purpose in real-life problems.

The first step of solving the trip planning task - data gathering - might affect significantly the chosen method of the following path finding step. The approach Photo2Trip [2] uses geo-tagged photos to extract interesting places on the map with an appropriate profit estimation for the target user. Results of this approach show that it is successful in several popular areas. On the other hand, the interesting segments (paths of interest) are not so well identified from photos. Another approach [3] uses a rich database of driving,

cycling or hiking trajectories to find the most popular route. The emphasis is on the creation of the network with transfer probabilities between nodes. To a great extent, this approach also depends on the quality of data from which the network is derived. Moreover, user interactions and possibilites of some specific, additional requirements and constraints are limited.

Apart from these approaches, other successful methods exist and according to their common features can be divided into these groups:

- Optimization based approaches
- Interactive based approaches
- Expert system based approaches
- Data mining based approaches

2.1.1 Optimization based approaches

One of the most common approaches is looking at the trip planning task as at an optimization task solving the **Orienteering Problem** (OP) [4]. Many exact and heuristic algorithms have been developed to solve some specific versions of the orienteering problem applied to the trip planning. The main idea is to transform the real map data input to a simplified model corresponding to an instance of the orieteering problem and then use an algorithmical approach to solve this instance. I have also used this approach and designed an algorithm for the modified version of the basic orienteering problem defined in the following chapter.

Approaches solving the OP can be divided into two groups depending on whether they guarantee an optimal solution. Solvers which find the optimal solution are usually based on the linear programming (e.g. [5]), the dynamic programming (e.g. [6]) or branch and cut techniques (e.g. [7], [8]). The disadvantage of these solvers is the execution time which grows almost exponentially with the number of input locations, and these solvers are not able to process hundreds of locations within a few seconds.

The paper [6] focuses on customized tour recommendations in urban areas described as the *TourRec* problem, which is a variation of the OP with additional constraints. Two algorithms were designed to solve two types of this problem. The first algorithm - *AdditiveTour* - maximizes the sum of profits from the visited locations, therefore it maximizes the overall quality of the tour. On the other hand, the second algorithm - *CoveringTour* - finds

diverse routes that offer a wide range of associated attractions and activities. Both algorithms use dynamic programming approaches which find optimal solutions but their execution time hardly depends on the number of input locations and number of input constraints (e.g. that one location must preceed another). According to the evaluation of these algorithms in [6], solutions (tours) are computed in real time for tour sizes up to 10 locations.

The other group of solvers contains approaches which do not guarantee finding the optimal solution and are usually based on some heuristics or approximation techniques. These solvers achieve better results for real-life problems related to the trip plannig since execution times are much shorter and the quality of results is still satisfactory. See detailed analysis of these approaches in the Orienteering Problem section below.

2.1.2 Interactive based approaches

These types of planners (e.g. Trip@dvice [9], Negotiation with a Software Travel Agent [10], INteractive TouRist Information GUidE (INTRIGUE) [11]) are based on gradual improvements of the initial solution according to user-added additional goals and constraints. Typically, the user defines some initial preferences and goals of the trip planning, after that the system generates an initial feasible solution and provides the user with this solution. Then the user can accept this solution or add/modify some constraints or goals and request the recalculation. The system generates another feasible solution respecting all constraints. These systems are partly dependent on the user's experience with the trip planning and it is more time-consuming to produce a trip itinerary which the user is satisfied with. The system designed in this thesis provides efficient recalculations which allow to use this system as an interactive based trip planner.

2.1.3 Expert system based approaches

Since the trip planning is a sort of a recommendation system for trips, several expert systems have been developed to solve the trip planning task. These systems often provide a variety of requirements defined by users. The approach evaluated in [12] searches trips according to the user's interest in specific activities (e.g. historic sites, outdoor activities, nature). These systems are usually mainly focused on some specific map domains, e.g. an expert system described in [13] has been developed specifically to solve the problem for 5

cities in Belgium. This thesis focuses on developing a general trip planning system for any map input, therefore an expert system is not preferred solution because of data demands.

2.1.4 Data mining based approaches

These planners use algorithms based on data mining which relies on well-described and rich map databases with scored locations and paths. These methods such as the Trip-Mine [14] or mining from trajectories [15] provide very good results, but the demand for a good data source does not allow to use this approach on the whole Earth map input, since some unpopular (but still interesting) parts of the Earth are very sparsely documented to be usable for this approach.

2.2 Map domains

Many complex systems have been created to solve or at least simplify trip planning tasks. Nevertheless, the majority of these systems is specialized for a specific map domain or works only under some hard constraints, not suitable for all real-life problems.

The general trip planning task for an arbitrary map input, each travel method and each optional constraint is very hard to define. Therefore, the trip planning is often divided into some groups of planning tasks where each group is somehow specialized for a specific domain of usage.

One of the most popular trip planning domains is the **city trip planning**. For these tasks, the typical element is a very dense map graph with a huge amount of roads and paths, sometimes in a multilayer perspective (e.g. navigation in a shopping center). These tasks also have a significant number of points of interest, typically with some additional constraints. The typical contraint of these points in the city planning tasks are for example time windows, meaning that each point of interest can be visited only in a specified time interval. Paths of interest, in this thesis called segments, do not occur so often in this planning domain.

The difference from the city trip planning is the **planning of routes in a landscape**. For these tasks, the typical map input is much less dense, but it is much larger. The number of points of interest is usually smaller than in the city trip planning but distances between them are much longer. We put more emphasis on paths we take between these points. We somehow have to distinguish between "nice" and "ugly" ways from the user's perspective. And also the distance between two locations in the landscape can vary a lot for two different users, much more than in the city planning tasks.

This paper focuses on the design of the general trip planning task solver, without restrictions on a map domain. But since the trip planning in a landscape requires more emphasis on segments, this is the main map domain I focused on and used for the testing.

2.3 Orienteering Problem

The general tourist trip planning task was described in [16] as the Tourist Trip Design Problem (TTDP) with some of its variations and extensions. The Orienteering Problem (OP) is the most basic version of the TTDP. Its generalization to a multiple-day trip planning is known as the Team OP (TOP) [17].

Many approaches have been proposed to solve the OP. This thesis combines several concepts of existing approaches to solve the modified version of the basic OP defined in the following chapter. Two main requirements for the OP solver for the planning task defined in this thesis are the **modifiability** and the **real-time processing**. The first requirement serves for the possibility of adapting the algorithm to the increasing number of constraints, since the trip planning is a complex task and many different improvements of the designed system might be added later. The possibility to include a new feature/constraint without losing the algorithms efficiency is essential. The second requirement reflects the demand for the real-time request-respond communication with the user. The system should be able to provide a trip planning solution within a few seconds to be effective and useful for the user.

The first solution for the OP was proposed by Tsiligirides in 1984 [18], where he described two heuristics (S-Algorithm and D-Algorithm) and evaluated their accuracy on several illustrative problems. Since then, these problems have become a good comparison tool for newer approaches. I have used some of these illustrative problems to test a part of my algorithm, see chapter 7.

Another approach presents the center-of-gravity heuristic [19]. The initial solution satisfying all OP constraints is constructed by the insertion heuristic. Then two steps are repeated: improving the solution by applying the cheapest insertion with the 2-OPT algorithm and the creation of a new route according to the sorted ranked locations with respect to the center of gravity. I have used the insertion heuristic concept for the creation of the initial solution in my algorithm.

An efficient four-phase heuristic to solve the OP was proposed in [20]. The four phases are termed *vertex insertion*, *cost improvement*, *vertex deletion* and *maximal insertions*. The first phase constructs an initial solution based on the relaxed travel budget constraint. The next phase improves the solution using the 2-Opt and 3-Opt algorithms from Lin and Kernighan [21] [22]. The third phase removes certain vertices from the tour in order to make room in the budget constraint. These three steps are repeated until a stopping criterion is met. After that, the final fourth phase improves the solution, attempting to insert as many of the remaining unvisited vertices in the sequence as possible. Results show that this heuristic finds near-optimal solutions within a few seconds. A similar solution proposed in [23] presents an effective heuristic which is also based on the construction of some feasible intital solution and then the modification of this solution using two-point exchange and one-point movement operations. The difference is that these phases are processed on the set of tours, which provides a richer exploration factor.

Another approach to solve the OP was proposed by Sevkli and Sevilgen [24] and uses techniques based on the Variable Neighborhood Search (VNS). This heuristic consists of four operations: *point insert*, *points exchange*, *sub route insert*, and *sub routes exchange*. These operations are then repeatedly applied starting from the initial solution in the form of an advanced local search procedure. I have used some parts of this concept for my trip planning solver designed in the following chapters.

Even the basic trip planning task is in practice slightly different from the OP. Two solutions of a trip planning task modelled as an OP might not be equally rated, even if their profit is the same. For example, the user might prefer the shorter solution. This requirement can somehow be included in the OP heuristic mechanism, but the second option is to use some post processing algorithms made for the Traveling Salesman Problem (TSP). One of them is the 2-OPT algorithm described and tested in [25]. Another option is to use some evolution approaches and use the tour length value as a criterion for the selection of better tours. I have used this approach in my system.

The approach by Gavalas [26] proposes the cluster-based heuristic for the Time Dependent Team Orienteering Problem with Time Windows (TD-

TOPTW) which can be used to model several real-life problems, especially useful for the public transport planning. The heuristic also uses the combination of operations such as shake, shift, replace and provides high quality results for up to 0,5 seconds considering topologies of 113 POIs.

2.4 Summary of used approaches

A basic trip planning task can be easily modelled as the Orienteering Problem (OP) [4] for which many solvers have been designed. The trip planning task with points and segments of interest defined in the following chapter can be modelled as an extended OP. The Tour Searching algorithm for the Automated Trip Planning System (ATPS) designed in this thesis, and solving this task, is based on famous optimization based heuristic solutions for the OP, the Variable Neighborhood Search (VNS) [24] algorithm, the insertion heuristic [19] and additional evolution approaches [27]. For the final evaluation of the system, an optimal algorithm is designed for the ATPS as an Integer Linear Programming (ILP) program based on the OP ILP solutions [5] and [28]. The ATPS is designed with focus on the Tour Searching algorithm which provides many additional features for the purpose of trip planning tasks in exchange for optimality. However, the optimal ILP algorithm can be incorporated into the system to produce optimal solutions of the defined task.

Chapter 3

Problem specification

In this chapter, I deduce simplifications of the real world planning mechanisms and then produce the definition of my trip planning task. This specification should work as an input for the automated trip planning system designed in the following chapter.

To find the required model of a trip planning process, let's ask a simple question: How do we plan a trip if we look at a map? Firstly, we propably have some prior knowledge about the surroundings, so we know where some interesting places are. In addition to this prior knowledge, we can extract some information from the map itself. Then we are designing a tour, trying not to break any of these basic rules:

1. We want to start and end at desired locations.
2. We want to finish the tour within an expected time span.
3. We want to visit the most interesting places.
4. We want to go along the most interesting paths.

The 1st and 2nd rule are simple not to violate if we know how to estimate the time demand. On the other hand, the 3rd and 4th rule are not so easy to guarantee, since we have to check every possible combination of visited places and paths in order not to forget anything. These are the steps which the automated trip planning system could help with. Any solution of the task

defined below will guarantee that rules 1 and 2 hold. The optimal solution will guarantee that rules 3 and 4 hold as well.

3.1 Map model

The main input for the planning task are map data. Many approaches have been developed for map representations. Since the trip planning is a sort of a map navigation task with some specific constraints and goals, I use the classical graph representation with nodes corresponding to GPS locations and edges representing traversable ways connecting these GPS locations.

Definition 3.1 (Node)
Let v be an arbitrary place on the Earth identified by the latitude(ϕ) and the longitude(λ), then v is called a node.

The node is in practice also characterized by a list of optional tags which describe this location (e.g. viewpoint, mountain challet, hill peak etc.).

Definition 3.2 (Edge)
Let e be a traversable path between some pair of nodes, then e is called an edge.

Definition 3.3 (Map distance function)
Let e be an edge connecting two nodes with GPS coordinates (ϕ_1, λ_1) and (ϕ_2, λ_2), then l is a map distance function and its value is according to the Spherical Law of Cosines [29] calculated as follows:

$$l(e) = R \cdot \arccos(\sin(\phi_1) \cdot \sin(\phi_2) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\Delta\lambda))$$

where $R \approx 6371$ is the approximation of the Earth radius in km, ϕ_1 and ϕ_2 are the latitudes and $\Delta\lambda = \lambda_2 - \lambda_1$ is the difference of longitudes.

Definition 3.4 (Map)
Let map $M = (V, E, l)$ be a **connected multigraph with oriented weighted edges**, where V is a set of nodes, E is a set of oriented edges, l is a map distance function which assigns a nonnegative value to each edge equal to the distance between two GPS coordinates.

To model the real world using a map model as realistic as possible, but with enough simplicity to be usable in the planning task, the following requirements hold for the map:

- **M is oriented**: I allow existence of ways which can be traversed only in one direction, even when walking (e.g. one way cable car, secured

mountain paths, via feratas etc.). Two way paths are modeled as two one way paths.

- **M is multigraph**: I allow multiple way existence between two GPS coordinates (e.g. one corresponds to road, the second for a sidewalk, but both start and end at the same GPS coordinates).
- **M is connected**: There is no reason to allow selection of two mutually unreachable nodes, therefore I assume that the map is a connected graph.

For practical reasons of the trip planning task I introduce an enhanced map model where for each edge, a time travel duration value is assigned.

Definition 3.5 (Map with travel duration function)
Let M be a map, then M_d is a map with the travel duration function $d : E \to R_0^+$ which assigns a nonnegative value to each edge on the map. This duration represents the time of travel along this edge with respect to this travel duration function.

In practice, each travel duration function is identified by a travel method (e.g. hiking, cycling, ...) or a user group (e.g. athlete, senior, ...) which allows to adapt time traversals for the specific purpose of the planning task.

Definition 3.6 (Point of interest = Poi)
Let's say that a node v in the M_d is Poi, if it has a profit score $profit(v) \in R_0^+$ and a time duration $duration(v) \in R_0^+$. I denote by POI the set of all Poi nodes in the map.

The profit score expresses the attractiveness of this place (node). In other words, how much the user wants to visit this place.

The time duration expresses how long it takes to visit this place (node). In other words, how long it takes for the user to fully obtain the profit score value.

Definition 3.7 (Segment of interest = Soi)
Let Soi denote a path $s = (v_0, e_1, v_1, e_2, ..., e_k, v_k)$, where v_i, e_i are nodes and edges in the M_d, respectively, which has a profit score $profit(s) \in R_0^+$. I denote by SOI the set of all Soi paths (segments) on the map.

The profit score of a Soi expresses the attractiveness of the path. In other words, how much the user wants to go along this path.

The length of a Soi is calculated as the sum of lengths of edges in the path $length(s) = \sum_{i=1}^{k} l(e_i)$.

The travel duration of traversing a Soi is calculated as the sum of travel durations of edges in the Soi, $duration(s) = \sum_{i=1}^{k} d(e_i)$, according to the

chosen travel duration function d. The travel duration of traversing some part (subpath) of the Soi is calculated similarly, by omitting unvisited edges in the sum.

3.2 Trip planning task

The basic trip planning task with points of interest (POI) and a time budget can be modeled as the Orienteering Problem [4]. I propose an extended trip planning task including segments of interest (SOI) which can better describe several real-life problems related to the tourist trip planning.

Definition 3.8 (Trip planning task with POI and SOI)
Let $U = (M_d, v_s, v_g, POI, SOI, B)$ be a trip planning task with points and segments of interest, where

- M_d is a map with a travel duration function d,
- $v_s \in V(M_d)$ is a start node of the trip,
- $v_g \in V(M_d)$ is a goal node of the trip,
- $POI \subseteq V(M_d)$ is a set of points of interest,
- SOI is a set of segments of interest (paths on the map M_d),
- $B \in R_0^+$ is a time budget of the trip.

Definition 3.9 (Tour)
A tour $T = (v_s, e_0, v_1, e_2, v_2, ..., v_k, e_k, v_g)$ is a path on the map M_d from v_s to v_g. I denote by $V(T)$ and $E(T)$ all nodes and edges in the tour T, respectively.

Definition 3.10 (Feasible tour)
The tour T is feasible if

$$\sum_{e \in E(T)} d(e) + \sum_{v \in POI \cap V(T)} duration(v) \leq B$$

Definition 3.11 (Optimal tour)
The optimal tour T^* is a feasible tour for which

$$T^* = arg\max_{T} \sum_{s \in SOI \wedge s \subseteq T} profit(s) + \sum_{v \in POI \cap V(T)} profit(v)$$

The $s \subseteq T$ relation between a segment s and a tour T is a hard constraint, which allows to gain profit only if the segment s is a subgraph of the tour T.

In other words, it is completely included in the tour T. But for the purpose of better modelling real-life problems, I have modified this constraint in the following chapter to gain profit according to the size of the segment included in the tour.

Definition 3.12 (Task solution)
A solution of the task U is any feasible tour T. The optimal solution of the task U is the optimal tour T^*.

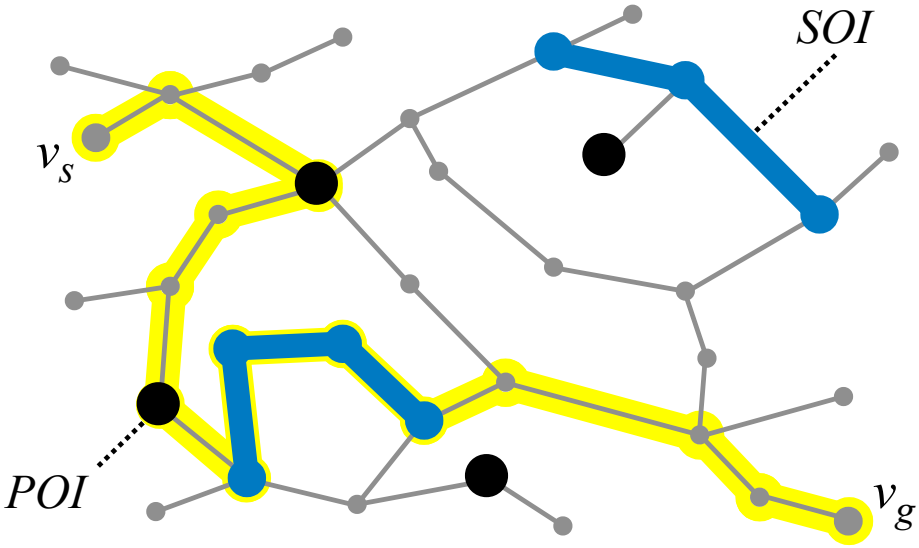


Figure 3.1: Visualization of an example instance of the trip planning task with POI and SOI. Nodes and edges in this graph represent M_d. A possible tour for this task is highlighted in yellow.

Chapter 4

System design

In this chapter, I design the Automated Trip Planning System (ATPS) and the Tour Searching (TS) algorithm which is the key part of the system. I combine several concepts described in the related work chapter and present a new approach which allows to process points of interest (POI) and segments of interest (SOI) together. The designed ATPS solves the task defined in the previous chapter and it can be used as an effective trip recommendation system.

I introduce several main requirements for the ATPS which also characterize the functionality of the system.

1. The input for the ATPS is the trip planning task with POI and SOI.
2. The output of the ATPS is a set of the best distinct feasible tours.
3. The ATPS provides results for a given tour request in a few seconds.
4. The ATPS provides fast recalculations for simple profit or time budget updates.

The first requirement ensures that the ATPS is usable for any general trip planning task with POI and SOI with the format described in the problem specification chapter. The second requirement defines the main goal of the trip planning task which is to generate a feasible tour (= trip which satisfies the user's demands). As an addition, the ATPS provides not only a single tour for one tour request, but it provides a set of the best distinct feasible

tours found. It means that the user has a pool of tours to choose from. The third requirement ensures that the ATPS is usable for practical trip planning applications when the user wants to get results for given trip requirements almost in a real time. The fourth requirement ensures that the user has the possibility to modify the profit of any point or segment of interest, or change the time budget, and request an even faster tour recalculation. The purpose of the ATPS is to provide an effective and interactive platform which recommends a variety of trips according to the user's preferences. The ATPS assumes that the user expresses their preferences with an input tour request and additional profit updates for specific *Poi* or *Soi*.

4.1 Solution overview

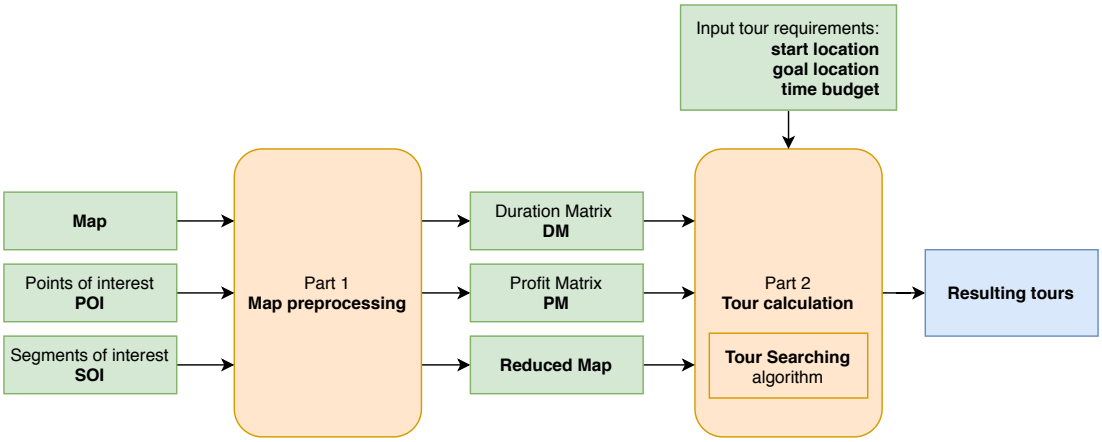


Figure 4.1: The general overview of the ATPS.

The general overview of the ATPS is described by the figure 4.1. The trip planning process of the ATPS consists of two main parts. The **Map preprocessing** part provides conversion of general map data with obtained sets POI and SOI into a simplified reduced map additionaly described by two matrices: the duration matrix and the profit matrix. The **Tour calculation** part is based on the Tour Searching algorithm which is executed on this reduced map to find the best feasible tours for specific input requirements. The main reason to split the trip planning process into these two parts is in efficiency savings. Each part is designed with respect to the expected frequency of executions. I assume that the map preprocessing is executed rarely, only when a new map data are needed. On the other hand, the tour calculation is executed frequently with every tour request from the user. This mechanism allows to pre-compute some data structures which are efficiently used by the Tour Searching algorithm.

4.2 Map preprocessing

This part of the ATPS provides transformation of general map data into simplified data structures used as an input for the tour calculation processes. The figure 4.2 shows the general workflow performed in this part.

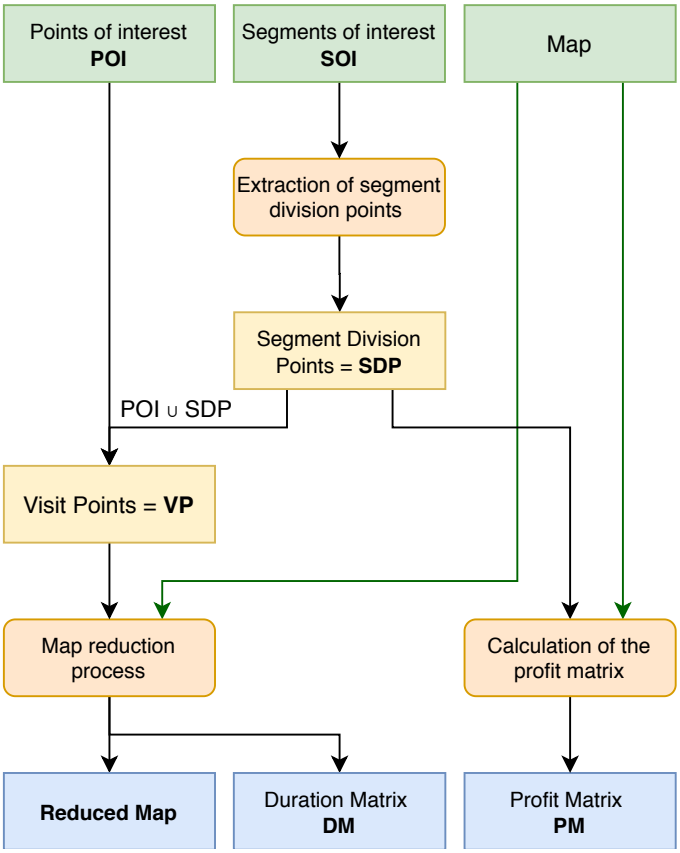


Figure 4.2: The workflow of the map preprocessing.

The main input for the map preprocessing is any map with the format of M_d defined in the problem specification chapter. Another inputs are the set of points of interest POI and the set of segments of interest SOI also with the format defined in the chapter 3. The output are three data structures: a simplified map model (called the reduced map) and two matrices which additionally describe some relations in this map. Now follows the detailed description of all processes needed to accomplish this transformation.

4.2.1 Extraction of segment division points

This process takes as an input all individual segments from the SOI and extracts specific nodes which are relevant for the Tour Searching algorithm. These nodes from segments correspond to turning points on the map. In other words, these nodes are important from the perspective of a planning task since a traveler might change the direction in this location.

Definition 4.1 (Segment Division Points = SDP)
Let SDP be a set of nodes from segments $s \in SOI$ which were selected by an extraction strategy.

Three different strategies can be used to solve this extraction problem based on input conditions:

1. If the whole segment has to be traversed to obtain its profit value, then only two segment division points are extracted: the start and the end node of the segment path.

2. If we allow a traveler to join and leave a segment at any location and obtain some part of the segment profit, then segment division points are all nodes on the segment which are crossroads. Meaning that more than two edges are incident with this node in the graph.

3. If we allow a traveler to join, leave and turn back at any location on the segment and obtain some part of the segment profit, then segment division points are all nodes on the segment.

The first strategy is very strict and does not desrcibe real-world situations very well. I do not use this strategy despite the fact that it simplifies the tour calculation process because of a smaller number of points considered.

The third strategy models perfectly the real world but the number of points considered hardly depends on lengths of segments. With long segment paths, the tour calculation becomes easily untractable because of the large search space.

The second strategy is somewhere between these two variants. For an average map input, the size of SDP is much smaller than using the third option, but it allows to find much more realistic tours on the map because of the possibility to join/leave the segment.

4.2.2 Map reduction process

This process creates a simplified map model by ommitting all nodes which are irrelevant from the trip planning perspective. This simple model is an abstraction of the original map and it is called the reduced map.

Definition 4.2 (Visit Points = VP)
$VP = POI \cup SDP$

Definition 4.3 (Reduced Map)
The reduced map of M_d is a complete graph with oriented weighted edges for which the following holds:

- Nodes are visit points (VP) of M_d.
- An oriented edge from a node i to a node j on the reduced map corresponds to the shortest path between these two nodes on the original map M_d with respect to the map duration function d. The length of this edge is equal to the length of this path in the original map M_d.
- For any path on the reduced map a corresponding shortest path on the original map exists.

The construction of the reduced map is done by computing shortest paths for every pair of nodes in the VP. The shortest path from a node i to a node j is computed using the A* algorithm. The admissible heuristic is chosen as the expected travel duration bewteen two GPS coordinates using the map duration function d. The time complexity is in the worst case $|VP|^2 \cdot O(|E| + |V|log|V|)$. Since the size of VP is much smaller than the number of nodes $|V|$ on the map, other approaches such as the Floyd Warshall algorithm are not feasible because of the time complexity $O(|V|^3)$.

The creation of the reduced map may be affected by additional trip requirements, such as ignoring some nodes or edges with specific tag (e.g. ignore all edges which are not bike paths). A specific reduced map is created for each special trip condidition. The shortest paths are used because I assume that from two different paths with same profits the user prefers the shorter one.

Definition 4.4 (Duration Matrix = DM)
Let M_d be a map and VP be a set of visit points in this map. Let $DM = |VP| \times |VP|$ be a matrix, where element $DM_{i,j}$ is the duration of traversing the shortest path from a node i to a node j on the map M_d with respect to the map duration function d. Elements $DM_{i,i}$ are set to zero. DM is called the duration matrix.

The duration matrix (DM) can be efficiently computed within the construction of the reduced map. We just sum up all edge duration values along the shortest paths found for each pair of nodes.

4.2.3 Calculation of the profit matrix

The profit value obtained by traversing some segment on the map has to be captured on the reduced map. A data structure called the profit matrix is designed to define the profit value obtained by traversing an edge on the reduced map.

Definition 4.5 (Profit Matrix = PM)
Let M_d be a map and VP be a set of visit points on this map. Let $PM = |VP| \times |VP|$ be a matrix, where element $PM_{i,j}$ is the profit obtained by traversing the shortest path from a node i to a node j on the map M_d with respect to the map duration function d without visiting any other node from VP. Elements $PM_{i,i}$ are set to zero. PM is called the profit matrix.

The construction of the profit matrix (PM) is done by following the shortest path for each pair of nodes in the VP and evaluating the profit obtained from traversing segments from the SOI.

I assume that if a tourist visits a part of a segment, the profit obtained from this segment should not be zero just because he missed another part of this segment. From this assumption, I approximate the profit obtained from traversing a subsegment s' of a full segment s with the following formula:

$$profit(s') = \frac{length(s')}{length(s)} \cdot profit(s)$$

This equation satisfies the condition that if a tourist visits the whole segment, they get the full profit value from this segment.

A special case can occur if the shortest path connecting two adjacent segment division points is not a part of this segment. In this case, a new segment division point is added in the middle of the segment path. It repeats until the shortest paths between all adjacent segment division points are also the paths on the segment.

4.3 Tour calculation

This part of the ATPS performs major calculations which find the most profitable tour satisfying user-input requirements. It is designed to provide good quality results for any tour request within a few seconds. The **Tour Searching** algorithm is the key component of this process. As an extension, the algorithm is designed to produce also additional top K distinct tours so that it easily provides a pool of different best trips for the user. The figure 4.3 describes the whole workflow.

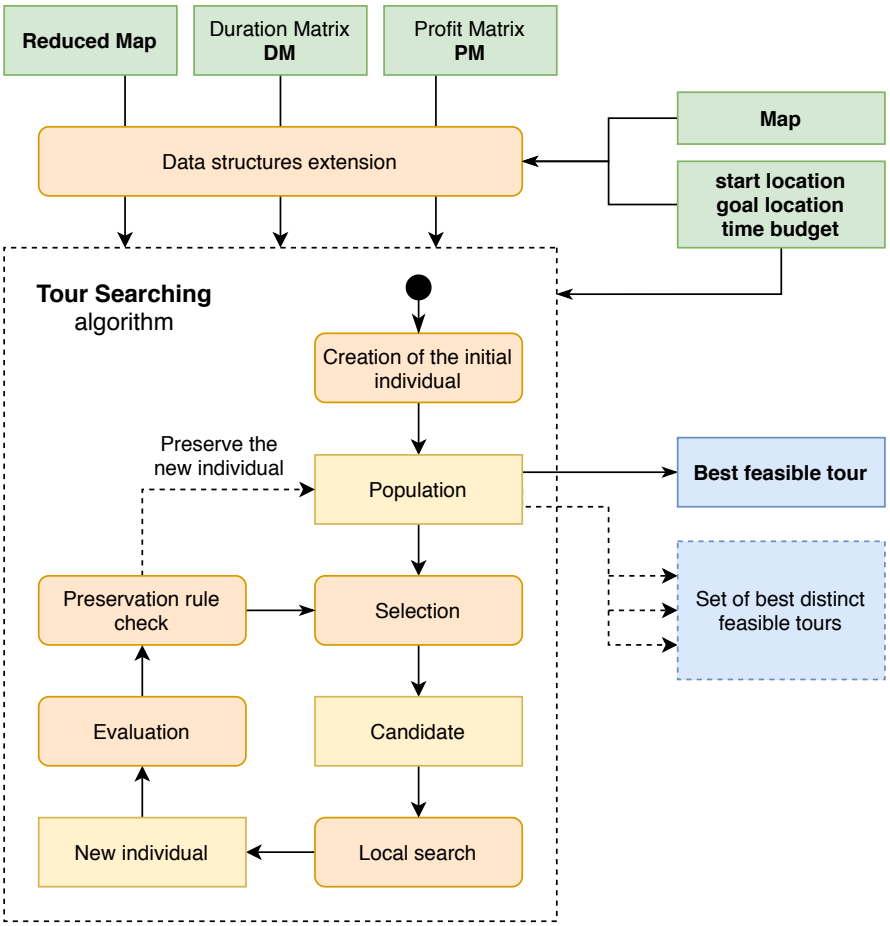


Figure 4.3: The workflow of the Tour calculation and the Tour Searching algorithm.

The main input for this process are three data structures created in the previous part (a reduced map, DM, PM) and user tour requirements (a start and a goal GPS location, a time budget). Before the Tour Searching algorithm is executed, these data structures are extended by two special visit points representing the start and the goal GPS location. For this extension, the original map is also needed as an input to place these points correctly. After

that, an initial feasible solution is created and the Tour Searching algorithm cycle starts. When the end condition is met, the searching process is stopped and a set of the best distinct feasible tours is obtained containing the best feasible tour found so far.

4.3.1 Data structures extension

To find a tour from a specified start and a goal location, a simple extension has to be made to the data structures from the map preprocessing part of the ATPS. These extended data structures work as an input for the Tour Searching algorithm. Two new visit points are created for the start and the goal GPS coordinates provided by the user. These two nodes are then incorporated to the reduced map and to both matrices.

The following operations are made to create the extended data structures:

- Two new visit points are created for the start and the goal GPS coordinates provided by the user.
- All shortest paths in the M_d are computed from the start visit point to all other visit points in the reduced map.
- All shortest paths in the M_d are computed to the goal visit point from all other visit points in the reduced map.
- The start and the goal visit points are added to the reduced map connected with all other visit points.
- The DM is extended by two rows and two columns for the start and the goal visit point with durations equal to the duration of computed shortest paths.
- The PM is extended by two rows and two columns for the start and the goal visit point with zero values since no segments are defined from these nodes.

A special case where the start location is the same as the goal location is modelled as two nodes with the same GPS coordinates. From now on, I assume that the reduced map, DM and PM are extended by the start and the goal node.

4.3.2 Tour Searching algorithm

The Tour Searching algorithm (TS) uses evolutionary concepts combined with basic ideas of the VNS algorithm [24]. It provides a mechanism to search efficiently the whole state space of possible feasible tours to find the most profitable tours. In the evolutionary naming conventions, this process of searching is also called the breeding of individuals. I design an individual which describes an arbitrary feasible tour. A set of these individuals is called a population. The whole process of the TS algorithm is described by the algorithm 1. The following sections describe all elements and procedures used in this algorithm.

Algorithm 1: Tour Searching

```
Input: E - ext. data structures, B - time budget, calculation time
Output: T - set of tours
u ← construct the initial individual
P ← ⟨u⟩ // the initial population
while calculation time not exceeded do
    c ← select a candidate from P
    s ← select a random shake operation (1, 2 or 3)
    l ← 1
    l_max ← 3 // maximum number of shake operations
    while l ≤ l_max do
        c' ← shake the c using the operation s
        c'' ← modify the c' by local search operations
        if the candidate c'' should be preserved then
            P ← insert the candidate c'' to the P
            l ← 1
        else
            s ← select an unused shake operation
            l ← l + 1
        end
    end
end
T ← extract tours from P
return T
```

The TS algorithm starts with the creation of the initial individual (see algorithm 2). Then the evolution loop is performed as follows: A candidate individual is selected from the current population according to the selection rule. This candidate is modified using shaking and local search operations based on the VNS algorithm [24]. After these modifications are done a new individual is obtained. This individual is evaluated to find the profit value of the tour it represents. After that it is discarded or added to the

population according to the preservation rules. This process is repeated until the calculation time is not exceeded. When this is completed, the current population contains the best individuals found so far, therefore it represent the set of the best tours found so far.

4.3.3 Individual

Each individual represents a unique tour on the reduced map for which a feasible tour on the map exists. Therefore each individual encodes a solution of the trip planning task with POI and SOI defined in the chapter 3.

Definition 4.6 (Individual)
An individual $I = \langle i_1, i_2, ..., i_w \rangle$ is a vector of w indices of visit points from the reduced map, where $0 \leq w \leq |VP| - 2$. I denote by i_s, i_g indices of the start and the goal node, respectively. These two nodes are never in the vector I.

Let T_k be the vector of the first k indices of visit points from the individual I, completed with the index of the start node i_s and the goal node i_g as follows

$$T_k = \langle i_s, i_1, i_2, ..., i_k, i_g \rangle$$

then the vector T_k represents a tour on the reduced map starting at the node i_s, visiting nodes $i_1, i_2, ..., i_k$ in this order and ending at the node i_g.

The duration of the tour T_k is calculated as the sum of durations between each adjoining pair of nodes plus the sum of durations of each visit point (zero for no Poi).

$$duration(T_k) = DM_{i_s,i_1} + \sum_{j=1}^{k-1} DM_{i_j,i_{j+1}} + \sum_{j=1}^{k} duration(VP_{i_j}) + DM_{i_k,i_g}$$

The profit of the tour T_k is calculated as the sum of profits between each adjoining pair of nodes plus the sum of profits of each visit point (zero for no Poi).

$$profit(T_k) = \sum_{j=1}^{k-1} PM_{i_j,i_{j+1}} + \sum_{j=1}^{k} profit(VP_{i_j})$$

Let B be an input time budget and T_k, T_{k+1} be two tours so that these two conditions hold together

$$duration(T_k) \leq B \text{ and } duration(T_{k+1}) > B$$

then the tour T_k is called the most profitable feasible tour represented by the individual I. Since for each feasible tour on the reduced map a feasible tour on the map exists, there exists a feasible tour T'_k on the map which is represented by the individual I. For a special case where $duration(T_{|VP|-2}) \leq B$, the most profitable feasible tour is $T_{|VP|-2}$.

For an individual I I denote by $profit(I)$ and $duration(I)$ the profit and the duration of the maximal profitable feasible tour it represents.

4.3.4 Creation of the initial individual

The creation of the initial individual is based on the construction of an initial feasible tour using the greedy strategy, or in some papers called the insertion algorithm ([20], [19], [18]). Indices of visit points in this initial tour are added to the start of the individual vector I without the special indices i_s and i_g. All other indices of unvisited nodes are added to the end of this vector.

The creation of the initial feasible tour using the greedy strategy starts with the most basic tour which immediately connects the start and the goal node with the shortest path. This initial tour can be represented by an empty individual $I = \langle\rangle$. A simple check confirms that the duration of this tour is not greater than the time budget B, otherwise no feasible tour exists. After that, a set A of all other available nodes is made. This set is a subset of all nodes in the reduced map containing all reachable nodes. A node with an index j is reachable if $DM_{i_s,j} + DM_{j,i_g} \leq B$.

After that we repeatedly select a candidate node from the set A and insert its index to the individual vector I. This cycle ends if the set A is empty or if we exceed the time budget. A candidate node is selected by the proportion of the increased duration and the increased profit. Let $I = \langle i_1, i_2, ..., i_k \rangle$ be a current initial individual with k indices, c be an index of a node from the set A and $p, 1 \leq p \leq k$ be a position in the I. Then $I' = \langle i_1, i_2, ..., i_{p-1}, c, i_p, i_{p+1}, ..., i_k \rangle$ is an individual created by insertion of

the node index c to the position p. We find the node index c and the position p for which

$$arg\min_{c,p} = \frac{duration(I') - duration(I)}{profit(I') - profit(I)}$$

Then we check whether the tour represented by the whole extended individual vector does not exceed the time budget. If not, we update the initial individual I and continue with the loop, otherwise we stop the loop. After that, indices of all unused available nodes are added to the end of the constructed individual I. The algorithm 2 describes the whole process.

Algorithm 2: Construct the initial individual

```
Input: E - ext. data structures, B - time budget
Output: I - initial individual
if DM_{i_s,i_g} > B then
    return no individual exists
end
I ← ⟨⟩
A ← set of reachable nodes
while A not empty do
    c, p ← select a greedy candidate from A
    I' ← individual I with inserted node c at position p
    if duration(I') > B then
        break
    else
        I ← I'
    end
end
append remaining available nodes from A to I
return I
```

4.3.5 Population

Instead of evolving a single individual only, a set of several distinct individuals is evolved. This set of individuals is called the population. Using this approach, I reduce the possibility of being stuck in a local optimum and I increase the diversity of state space search. An additional benefit of this approach is that the algorithm is breeding several distinct individuals (tours) at the same time. The result of this calculation is not a single tour but a set of the best distinct tours. Two rules have to be designed to use this evolution approach. The **selection rule** defines how candidates are selected for local search

improvements and the **preservation rule** defines which evolved individuals are added and kept in the population.

Definition 4.7 (Population)
Let P be an ordered sequence of $|P|$ individuals for which $profit(P_i) \geq profit(P_{i+1}), 1 \leq i < |P|$ holds, then P is called the population. I denote by the population limit the maximum number of individuals which can be in the population.

Selection rule

The Roulette-wheel selection [30] is used to select an individual from the population. Using this method, I favor individuals with a greater profit value but I keep nonzero probability of selecting a less profitable individual to extend the state space exploration.

Let P be a population of $|P|$ individuals. The probability of selecting an individual i is calculated as follows:

$$p_i = \frac{profit(P_i)}{\sum_{j=1}^{|P|} profit(P_j)}$$

Preservation rule

The preservation rule defines which candidates are kept in the population. Since my starting population is just a single individual (found by the algorithm 2), this rule mainly defines the process of adding a new individual to the population. For the purpose of avoiding being stuck in a local optimum I keep only distinct individuals in the population. By distinct I mean that they are not similar to each other.

Definition 4.8 (Similarity of individuals)
Let I_1, I_2 be two individuals and T_{k_1}, T_{k_2} tours which they represent. Let S be the set difference of indices of nodes in these tours and $R \in \langle 0, 1 \rangle$ be the similarity constant. Then if $|S| < R \cdot max(k_1, k_2)$ I denote these two individuals as similar.

The process of adding a new individual I to the population proceeds as follows:

1. Determine if a similar individual I_s already exists in the current population.
   - If I_s exists and $profit(I) > profit(I_s)$, then replace I_s by I in the population.
   - If I_s exists and $profit(I) = profit(I_s)$ and $length(I) < length(I_s)$, then replace I_s by I in the population.
   - If I_s does not exist, then insert I to the appropriate position in the population according to the profit value. The best individual is always kept on the top position.
2. If the population size is over the limit, remove the least profitable individual.

4.3.6 Operations

Modifications made to the candidate individual are of two kinds: shake and local search operations. Two shake operations are: *move subsection* and *swap two subsections*. Since these operations perform wider changes to the individual, I consider a special third shake operation which does no change to the individual and allows to explore the nearest neighbors. Local search operations are divided into two group. Uninformed operations: *move one node*, *swap two nodes*, and informed operations: *smart move of one node*, *smart move of a successor*. These operations slightly modify an individual to perform uninformed or informed exploration of the nearest neighbors. Here are the details of how each operation is performed:

- Shake operations
  1. Move subsection

     A randomly chosen subsection is moved to a new position in the individual. Using three random indices $1 \leq j_1 < j_2 < j_3 \leq |I|$, a modification is made by moving the sequence of nodes at positions $l, j_1 \leq l \leq j_2$ after the position j_3. Alternatively, with the same probability, the sequence of nodes at positions $l, j_2 \leq l \leq j_3$ is moved before the position j_1.
  2. Swap two subsections

     Two randomly chosen subsections are swaped in the individual. Using four random indices $1 \leq j_1 < j_2 < j_3 < j_4 \leq |I|$, a modification is made by exchanging the sequence of nodes at positions $l, j_1 \leq l \leq j_2$ with the sequence $h, j_3 \leq h \leq j_4$.

3. No shake

   Shake is not performed. No change to the individual.

- Local Search operations

  1. Move one node (uninformed)

     A randomly chosen node is moved to a new position in the individual. Using two random indices $1 \leq j_1, j_2 \leq |I|, j_1 \neq j_2$, a single modification is made by moving a node at the position j_1 after a node at the position j_2.

  2. Swap two nodes (uninformed)

     Two randomly chosen nodes are swapped in the individual. Using two random indices $1 \leq j_1, j_2 \leq |I|, j_1 \neq j_2$, a single modification is made by exchanging a node at the position j_1 with a node at the position j_2.

  3. Smart move of one node (informed)

     A randomly chosen node is moved to the position where the extension in the duration between the neighboring nodes is the minimal. A random index $1 \leq j_1 \leq |I|$ chooses the node to move, after that the position j_2 is found for which the insertion of this node would lead to the minimal increase of the tour duration.

  4. Smart move of a successor (informed)

     For a randomly chosen segment node a missing successor is appended. A segment node with the highest index less than a randomly generated index $1 \leq j_1 \leq |I|$ is found. If it exists, the most profitable segment successor is inserted after this node in the vector. Alternatively, the most profitable predecessor is inserted before this node.

The initial shake operation is selected at random and changed whenever the modified individual is not added to the population. This means that it is not better than any existing individual or it is similar to some individual in the population. The local search operations are performed in groups as described in the table 4.1. This approach with the selected probabilities is based on the VNS from [24] and the effective heuristic designed in [23]. I have combined and extended these approaches with the smart (informed) variations of the operations which proved to be very efficient for this type of trip planning task.

Group	Operations	Probability
uninformed	move one node swap two nodes	0,5
informed	smart move of one node smart move of a successor	0,5

Table 4.1: Operations performed during the local search procedure.

4.4 Recalculation

The evolution approach used in my algorithm provides the possibility of the calculation continuation. If another tour request is obtained and this request has the same start and goal positions as the previous tour request, the new calculation can be based on the previous results, therefore, it can produce good quality results even in a shorter calculation time. This mechanism allows to use the ATPS as an interactive platform for the user to adjust some additional trip preferences. The diagram 4.4 shows the general workflow of the recalculation process.

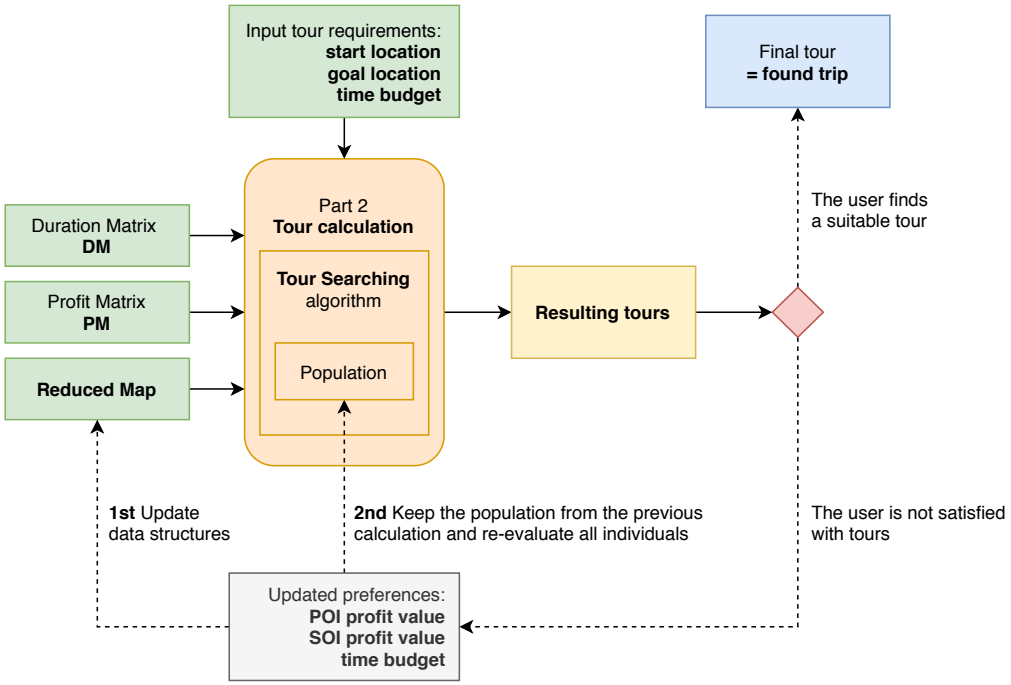


Figure 4.4: The general use case workflow of the recalculation process.

The calculation continuation can be used if the time budget has changed or profit values of some Poi or Soi have changed. It also can be used if no update has been made and we just want to increase the execution time of the previous calculation. The recalculation based on these updated values is performed in two steps.

Firstly, the data structures are updated based on new profit values. If the profit value of a Poi has changed, the appropriate profit value is updated in the reduced map for this Poi. This modification is performed in a constant time. If the profit value of a Soi has changed, corresponding edge profits in

the PM are updated according to this new profit value. This modification takes at most $O(n^2)$ time with respect to the number of SDP in the updated segment.

Secondly, the population from the previous calculation is used as a baseline for the new tour calculation process. Since profits of individuals in this population might changed, all these individuals are re-evaluated and sorted according to their new profit value. Also, if the time budget has changed, all tours of individuals in the population are extended or truncated (based on the extension or shortening of the time budget), and all individuals are re-evaluated and sorted according to their new profit value again.

After these steps are finished, the Tour Searching algorithm is executed starting with this created population. The initial individual is constructed and inserted to the population according to the preservation rules and the evolution loop is started.

This mechanism assumes that the resulting tours for the updated task do not very differ from the original task results. Therefore, we use previous results as a baseline to efficiently spend the recalculation time on even deeper exploration of possible better solutions.

Chapter 5

Optimal algorithm

In this chapter, I design the Basic ILP optimal algorithm to solve the trip planning task with POI and SOI, mainly due to the evaluation of results produced by the Tour Searching algorithm.

The TS algorithm designed in the previous chapter is a heuristic approach which does not guarantee finding the optimal solution, but it provides many additional benefits. Especially, it produces K distinct tours within a single tour request, it guarantees a feasible solution at any point of time and it processes huge input maps in a short amount of time. These are the benefits which could not be achieved using current optimal approaches. On the other hand, this optimal algorithm may work as a complement for the ATPS to guarantee solving some small task instances optimally.

5.1 ILP formulation

I propose an Integer Linear Programming (ILP) formulation for the trip planning task with POI and SOI. Formulations were created based on the ILP solutions for the OP [5] and the Miller–Tucker–Zemlin Subtour Elimination Constraints [28]. This solution assumes that the general task is transformed into data structures according to the map preprocessing process of the ATPS and that these data structures are extended by a start and a goal node.

The following program assumes that the reduced map, the duration matrix (DM) and the profit matrix (PM) were constructed containing N visit points. The start visit point has the index 1 and the goal visit point has the index N.

$$\max \sum_{i=1}^{N-1} \sum_{j=2}^{N} x_{ij}(profit(VP_i) + PM_{ij}) \tag{1}$$

$$\sum_{i=2}^{N} x_{1i} = \sum_{j=1}^{N-1} x_{jN} = 1 \tag{2}$$

$$\sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1; \ \forall k = 2, ..., N-1 \tag{3}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} x_{ij}(duration(VP_i) + DM_{ij}) \leq B \tag{4}$$

$$u_i - u_j + 1 \leq (N-1)(1 - x_{ij}); \ \forall i, j = 2, ..., N \tag{5}$$

$$2 \leq u_i \leq N; \ \forall i = 2, ..., N \tag{6}$$

$$x_{ij} \in 0, 1; \ \forall i, j = 1, ..., N \tag{7}$$

A binary variable x_{ij} is created for each edge in the reduced map, the equation (7), N^2 variables in total. The value x_{ij} equal to 1 means that an edge from a node i to a node j is used in the tour, the value is equal to 0 otherwise. The constraint (2) ensures that exactly one edge incident with the start and the goal node is used, respectively. The constraint (3) provides the flow controll, meaning that for each node the number of incoming and outcoming edges must be equal, and not greater than 1. The flow controll is not considered for the start and the goal node. The constraint (4) ensures that the duration of the tour is not greater than the time budget B. Constraints (5) and (6) are the Miller–Tucker–Zemlin Subtour Elimination Constraints [28] to remove potential inner loops. The objective (1) maximizes the overall profit value of the tour which is the sum of profits of individual Poi and the sum of profits from the profit matrix (PM) representing profits of individual Soi.

The final optimal tour is decoded from variables $x_{ij} = 1$, representing used edges on the reduced map for which corresponding shortest paths on the original map exist.

Chapter 6

System implementation

In this chapter, I provide a brief description of all tools and techniques which were used to implement and test the ATPS designed in the previous chapters. Mainly I focus on two main parts of the whole realization process. Firstly, the data gathering including the data preprocessing techniques. Secondly, the implementation of the whole ATPS.

6.1 Data gathering

A reliable data source is one of the most important elements which affects the trip planning results. Good maps are the main database of every trip planning system. For ideal general trip planner, it should be as detailed as possible, often updated, and it should provide data for as many places on the Earth as possible. Many science groups and companies have created a massive number of different maps where each map is usually designed with emphasis on a specific purpose. The trip planning requires maps which capture especially reliable road systems and locations related to the tourism. Several commercial or open source solutions exist providing these maps. The technology called **Open Street Map** (OSM) [31] is an open database project which covers all these requirements, and additional tools are provided to work with these data easily for the purpose of the ATPS.

6.1.1 Open Street Maps

The **OSM** [31] is an open database (accessible here *www.openstreetmap.org*) of maps which cover the whole world with a detailed description of all locations which are frequently visited by tourists. It is frequently updated by the community of contributors from the public where each change is reviewed with respect to the impact on systems currently using this database.

This technology provides an export in the osm file format which is an xml file encoding the map graph with additional information about interesting locations. For a specific bounding box described by two latitudes and two longitudes, the **Overpass API** [32] generates the appropriate map file which is a cutout of the global map. This API has been used to generate all map files used for the testing of the ATPS.

6.1.2 Map file preprocessing

Since osm map files are provided in a rich form with all details, they are usually unnecessarily large. The Java tool **Osmosis** [33] provides several commands which manipulate the osm files so that these files can be easily merged and resized, and it also provides the filtration mechanism of unnecessary data.

The script **reduceOSMmap.bat** (available in the attachment) was created to automate the process of modification of raw osm files into reduced osm files containing only data which are required for the ATPS. In average, this filtration reduces the map graph size to 10 % of its original size.

A configuration file was designed to define sets POI and SOI in the osm map file. It is a text file with the cfg extension with the content of the followng format. Lines starting with POI define which nodes in the osm map file are points of interest according to their tag information. The profit values and the expected durations in minutes are set for these nodes. Lines starting with SOI define segments in the osm map file which are the shortest paths between two GPS coordinates. The profit values are set for these segments.

A template for the configuration file format:

```
POI <OSMTag> <OSMValue> <Profit> <DurationInMin>
SOI <SrcLng> <SrcLat> <DstLng> <DstLat> <Profit>
```

6.2 Implementation

In this section, I briefly describe the architecture and all technologies used for the implementation of the ATPS prototype. The final implementation contains all key components of the whole system which can be easily extended for a further deployment to solve specific real-life problems.

6.2.1 Architecture

My implementation of the ATPS is divided into three main layers: the client web application, the server and the C++ Addon with a map database. See the visualization of this architecture in the figure 6.1.

The topmost layer is the client web application written in the javascript representing the user interface for the ATPS. It provides basic map visualization using the MapBox API [34] and it enables to send trip requests and display obtained results.

The middle layer is the server web application written in the javascript using the NodeJS [35] technology. This layer controls the initialization of the ATPS, handles currently loaded maps, stores pre-calculated data and it provides communication between client side web applications and low level C++ Addon.

The third layer is the C++ Addon which contains algorithms for the main trip planning logic designed in the previous chapters. According to designed ATPS parts the implementation of this addon is divided into two main components: the map preprocessor and the trip planner.

During the implementation, I have mainly focused on the correct and efficient functionality of the C++ Addon which can be further incorporated into other systems using different technologies for server and frontend implementations. The first and the second layer of this system were created to test this addon properly and provide the idea of possible applications in real-world scenarios.

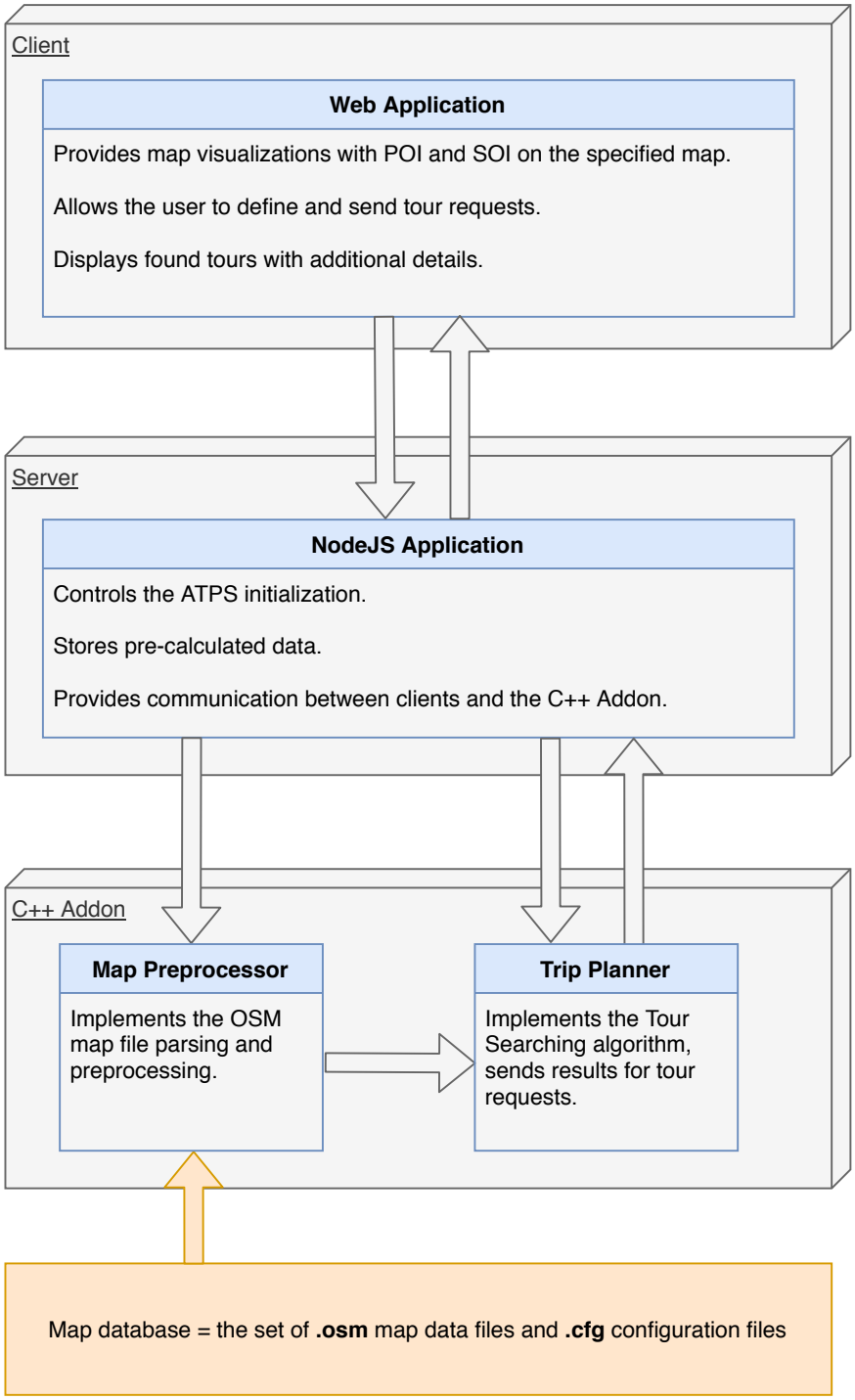


Figure 6.1: The high level architecture of the implemented ATPS.

6.2.2 User interface

The user interface implemented for the ATPS and represented by the client side web application contains several key functionalities. It provides the map visualization using the MapBox API [34] and it enables the user to send a tour request and display obtained results. Any tour request contains a start location of the trip, a goal location of the trip and a time budget (an expected trip duration) in hours.

As an addition for the testing purposes the user interface enables to adjust the calculation time of the Tour Searching algorithm so that the user can increase the basic calculation time for possibly better results. After the submission, the request data are propagated by the server application to the C++ Addon.

The typical use case is as follows:

1. The user selects an appropriate map from the map database. After the map is loaded it is displayed on the screen with all points and segments of interest. The user can view the detailed description of any Poi.

2. The user sets the start and the goal location of the trip using the drag and drop markers or buttons on the left panel.

3. The user changes the expected trip duration in hours.

4. The user can adjust the calculation time of the Tour Searching algorithm. The default value is 2 seconds.

5. The user clicks the Find Trip button to submit the tour request.

6. After the calculation finishes, all found trips are displayed on the left pannel sorted by the profit value. For each trip, the length and the duration of the tour is displayed. The user can hover the mouse over these trips and view the tour on the map.

7. If the user is not satisfied with the results, he can additionally adjust the profit value for any Poi using the star rating and the system will automatically provide updated results for this change.

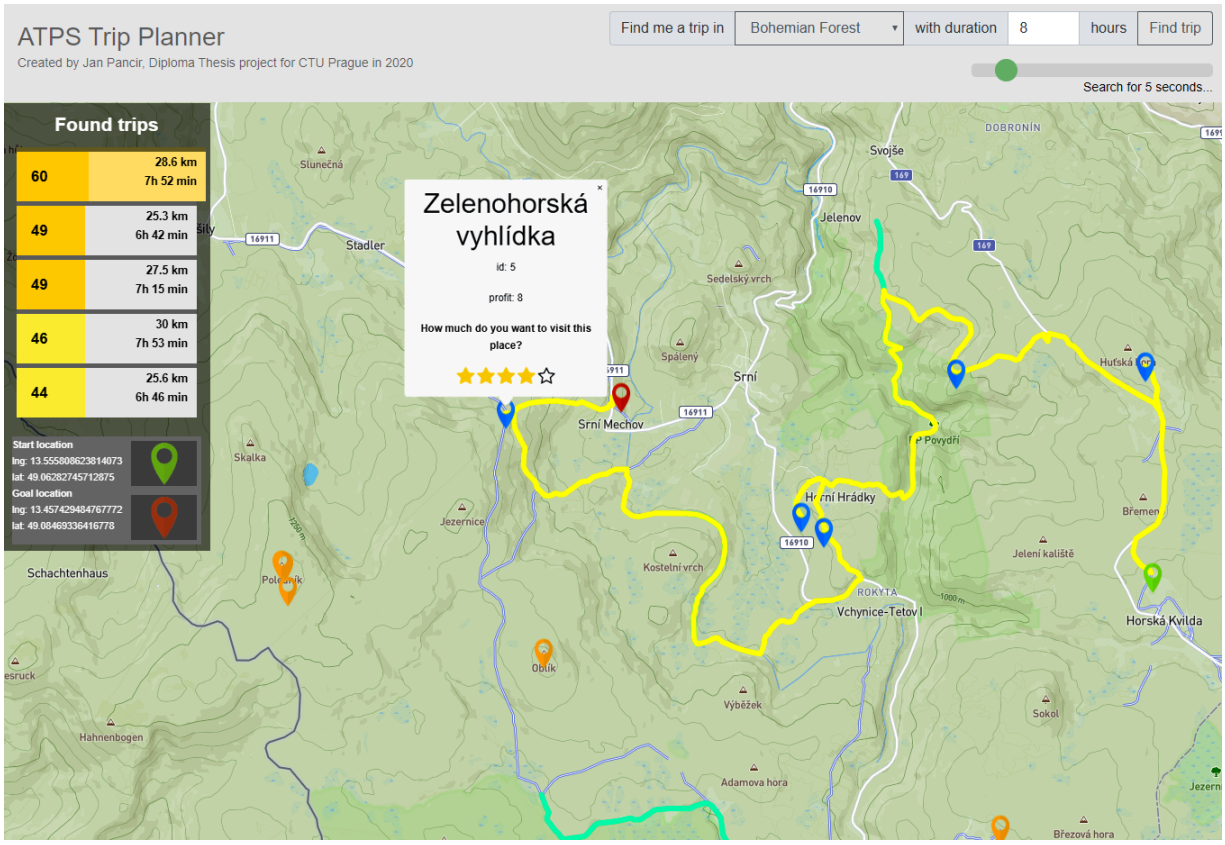


Figure 6.2: The user interface preview of the implemented ATPS. The map visualization contains visited Poi (blue markers), unvisited Poi (orange markers), Soi (light blue path) and the found tour (yellow path). The start and the goal positions are highlighted with green and red markers. Each Poi can be viewed in detail and a custom profit value can be set for it using the star rating.

6.2.3 C++ Addon

The third layer is an addon for the NodeJS [35] server which contains the key logic of the ATPS. It is written in C++ especially to effectively use the available computational power for time demanding operations. The functionality of this addon can be divided into two main components. The map preprocessor and the trip planner.

Map preprocessor

The map preprocessor provides the functionality to load and parse the osm map data file with its configuration file and create a graph representation of the map. The largest connected component is found in this graph using the DFS algorithm to eliminate unreachable subgraphs. After that it performs the whole map preprocessing part of the ATPS and creates data structures for the tour calculation process: the reduced map, the duration matrix and the profit matrix. These data structures are implemented as three 2D arrays.

Trip planner

The trip planner performs the whole tour calculation part of the ATPS. It implements the data structure extension and an algorithm which finds tours in these data structures. The ATPS is based on the Tour Searching heuristic algorithm, but the optimal ILP algorithm can be incorporated to the system to obtain optimal tours with loss of advantages of the heuristic algorithm.

The **Tour Searching** algorithm is implemented in the *OPSolver* class of the system. Each individual is endoded as an integer vector and the population is a vector of these individuals which can be replaced by a priority queue if a significant number of individuals is used. Multithreaded implementation allows to execute several runs of the Tour Searching algorithm at once and gather the best result from these runs. This solution eliminates majority of possible suboptimal results returned to the user because of the algorithm being stuck in a local optimum.

The **Basic ILP** optimal algorithm was implemented using the Gurobi optimizer [36] and is attached as a separate project which can be incorporated into the ATPS.

Chapter 7

Testing and results

The testing of the ATPS is performed using two different approaches. The first approach called the algorithm evaluation uses specially designed artificial input instances inspired by the traditional orienteering problem test benchmark. The goal of this testing is to compare results of the Tour Searching algorithm with optimal solutions from the Basic ILP algorithm. The second approach performs an overall testing of the ATPS on real map data with focus on real-life problems. In this chapter, both approaches are described and results obtained are presented.

All experiments were executed using the AMD Ryzen 5 3600X 6-Core Processor. The Gurobi optimizer for the Basic ILP algorithm was executed on 12 threads and the Tour Searching algorithm was executed on 8 threads for these experiments.

7.1 Algorithm evaluation

This testing approach is specially designed to test the main unit of the ATPS, the Tour Searching algorithm, compared with the optimal Basic ILP algorithm. This evaluation performs testing to confirm that the TS algorithm provides valid results of sufficient quality, the results are getting better with the increasing calculation time and also to measure how close results are to the optimum.

7.1.1 Custom test benchmark

For the purpose of this testing, a custom test benchmark set was created. Since my trip planning task with POI and SOI defined in the chapter 3 is an extension of the classic OP task [4], I have created sample test instances based on the OP benchmark instances for 32-locations problem created by Tsiligirides [18]. This original benchmark set corresponds to my task if no segments were considered. I have extended this 32-locations problem by 6 segments with different lengths and profits, so it contains 52 visit points in total. I refer to this test benchmark as 52-VisitPoints problem. The placement of points and segments of interest in this problem is shown in the figure 7.1. This new benchmark set should work as a comparison tool to verify that TS algoritm provides near optimal results in a very short calculation time. It also verifies that both my algorithms (TS, ILP) provide at least as good results as algorithms used to solve the original OP benchmark by Tsiligirides [18].

To perform this testing, the benchmark runner component was implemented for the ATPS. Using this runner, the Tour Searching algorithm or the Basic ILP algorithm can be easily executed to solve any problem described by the following file format:

```
# POI SOI Benchmark test file definition
<TimeBudget>
POI
<X> <Y> <PoiProfit>
...
SOI
<SoiNodesCount> <SoiProfit>
<X1> <Y1>
<X2> <Y2>
...
```

An euclidean distances are considered in this problem and the travel duration function d is equal to the map distance function l. For these test instances, no duration time is considered for points of interest. The test benchmark set for the 52-VisitPoints problem in this file format is available in the attachment.

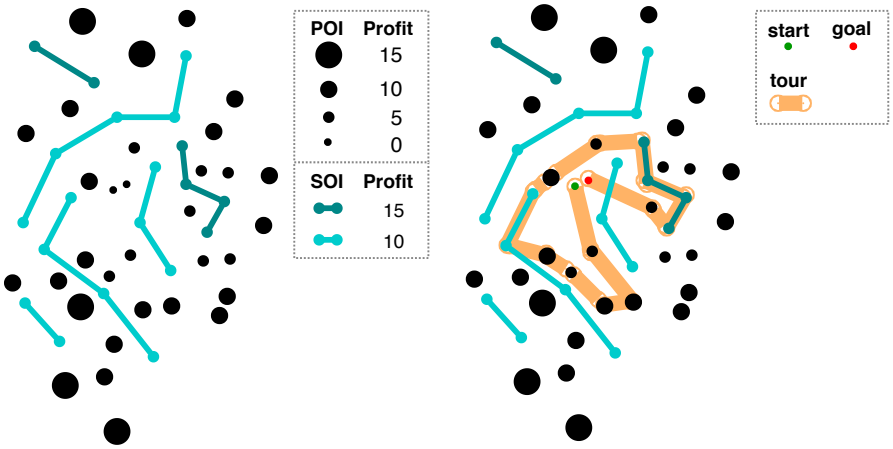


Figure 7.1: The visualization of the 52-VisitPoints problem. The left figure shows POI and SOI with their profits which correspond to the data structures format. The right figure shows a possible tour found by the Tour Searching or the Basic ILP algorithm for obtained start and goal nodes and a certain time budget.

7.1.2 Experiments and results on the test benchmark

The Basic ILP and the Tour Searching (TS) algorithms were executed to solve the 52-VisitPoints problem for several time budgets. The TS algorithm was set up with the population size 10, and the similarity constant $R = 1/3$, so it provides up to 10 distinct tours within a single execution, if distinct tours exist. The calculation time limit of TS algorithm was set to 30 seconds and each execution was repeated 10 times to obtain average results. The table 7.1 shows the results. For each time budget (B) the profit value (P_{opt}) and the length of the optimal tour (L_{opt}) was found by the basic ILP algorithm in the execution time T (in seconds). The T_{opt} is the average time (in seconds) when the TS algorithm found the optimal solution. Some values are missing since the TS algorithm got stuck in some cases, so no optimal solution is guaranteed. P_{2sec} and P_{30sec} are average profit values of best tours found by the TS algorithm in 2 and 30 seconds. Details to all tours found in this experiment are available in the attachment.

	Basic ILP			Tour Searching		
B	P_{opt}	L_{opt}	$T(sec)$	$T_{opt}(sec)$	P_{2sec}	P_{30sec}
5	10,00	4,14	0,03	**0,1**	**10**	**10**
10	15,00	7,95	4,22	**0,1**	**15**	**15**
15	45,00	14,26	9,89	**0,1**	**45**	**45**
20	65,00	19,60	115,95	**0,1**	**65**	**65**
25	90,00	24,82	175,80	**1,5**	**90**	**90**
30	110,00	29,88	628,25	**0,5**	**110**	**110**
35	135,00	34,84	163,32	**1,4**	130	**135**
40	155,00	39,48	262,48	**1,2**	155	**155**
46	180,00	45,86	22,99	**2,3**	175	**180**
50	195,00	49,34	22,05	**8,4**	190	**195**
55	210,00	53,93	52,88	**5,5**	200	**210**
60	225,00	59,76	39,16	**5,4**	220	**225**
65	245,00	64,86	20,84	**6.5**	230	**245**
70	265,00	69,59	4,11	-	240	250
73	275,00	72,90	7,20	-	250	255
75	285,00	74,72	1,93	-	260	285
80	300,00	79,17	0,77	-	280	290
85	310,49	84,73	0,91	-	300	305

Table 7.1: Results of the basic ILP and the TS algorithms on the 52-VisitPoints problem.

The search space of possible tours is the largest between time budgets from 20 to 65. The results show that the basic ILP algorithm finds optimal solutions for some of these cases in more than hundreds of seconds. The TS algorithm provides optimal or near optimal results in 2 seconds. The quality of obtained results from the TS algorithm increases with the calculation time, but in some cases the algorithm gets stuck in a local optimum and it does not guarantee finding the optimal solution. With very long time budgets, the Basic ILP algorithm overcomes the TS on this problem, since the search space is very sparse in these cases but the length of tours makes this task harded for the TS algorithm. However, the efficiency of the calculation is sufficient for the purpose of trip planning. It needs to be mentioned that for all best tours measured in this experiment, the Tour Searching algorithm provided up to 10 distinct tours. Thus, the real applicability and usability is definitely wider for these results with respect to the trip planning task. The conclusion is that the algorithm provides good quality tours in a few seconds on the 52-VisitPoints problem.

7.2 Real map testing

The testing of the designed ATPS was performed on real map data using four map areas gathered from the OSM database [31]. Several testing tour requests were performed to adjust parameters of the system, analyze the profit improvements per calculation time, analyze determinization of the results, and verify that tours found by the ATPS are usable in real life and that they are similar to manual tour recommendations created in these map areas.

7.2.1 Map input

Four areas from the Czech Republic were selected for the real world map data testing of the ATPS. The selected areas are: **the Giant Mountains, the Jesenik, the Bohemian Forest and the Bohemian Paradise**. These locations are frequently visited tourist places which are well mapped by the OSM database and they contain many tagged places which can be easily used as points of interest. Also these locations consist mainly of landscape areas for which the segments of interest are available. Segments are not so frequent in cities.

The map files for these areas were exported from the OSM database and reduced according to the data gathering section described in the previous chapter. After that, config files describing the POI and the SOI were created for each map file. The following description of POI was used:

```
# TAG | VALUE | PROFIT | DURATION (MIN)
POI tourism apline_hut 5 60
POI tourism viewpoint 8 10
POI tourism attraction 10 15
POI tourism picnic_site 2 10
```

Several segments of interest were manually created according to popular and frequently visited paths is the specific areas, mainly inspired by Mapy.cz [37] and Trip Advisor [38]. These segments should work as a basic demonstration of this feature which can be further extended.

Map	Nodes	POI	SOI	VP	Description
Giant Mountains	134732	257	10	360	large map size long SOI typical for long route trips
Jesenik	135840	158	0	158	large map size no SOI
Bohemian Forest	31661	78	5	121	medium map size sparse placement of VP
Bohemian Paradise	44854	71	13	163	medium map size typical for round trips

Table 7.2: The description of maps used for the real data testing.

Some details of these maps are described in the table 7.2. The largest map captures the whole Giant Mountains national park with the size around 40x30 km. This map contains the most visit points so the tour calculation task is the most difficult on this map. It was selected to test mainly the long route planning, e.g. multiple day hiking trips in a landscape. The protected landscape area Jesenik is the second map area which is almost the same size as the previous map. No segments were created for this map to test the differences between the trip planning on maps with and without segments. The other two maps, the Bohemian Forest and the Bohemian Paradise were selected to simulate the most probable use case of the trip planning. These maps are of a sufficient size and detail to plan multiple one day trips to these locations. The difference in these maps is that the Bohemian Paradise map is mainly focused on finding round trips.

The travel duration function d used for these maps was set fixed to approximate the walking speed of 3,4 km/h. This value was selected based on the measurements taken in [39].

7.2.2 Experiments with parameters

Experiments are performed to tune values of important parameters which affect results of the ATPS. Optimal values for the population limit and the similarity constant are found.

Population limit and K tours

The goal of this experiment is to find the optimal population limit and also the K tours value. The population size specifies the number of tours which can be effectively found during one tour request. The K tours value is the number of tours which are provided to the user. Since in the trip planning task we are looking for the most profitable tour in the first place, I select the population limit according to the profit of the best tour found. The additional found tours are a 'nice to have' feature of this algorithm to provide the pool of the best different tours for the user. But this feature should not greatly affect the profit of the best tour.

A sample tour request with the time budget 6 hours and the calculation time 5 seconds was executed 10 times on the Bohemian Paradise map. The average profit of the best tours found reached a maximum with the population limit around 10 individuals. The table 7.3 shows all results. For other experiments, I have used the population limit 10 and the K tours value 5 which is sufficient.

Population limit	1	2	5	**10**	20	40	80
Avg. Profit	116	117	121	**122**	120	120	115

Table 7.3: Average profits for a sample tour request on the Bohemian Paradise map depending on the population limit.

Similarity constant

The goal of this experiment is to find the optimal similarity constant R. It defines when we call two selected individuals (tours) similar. This constant affects how tours differ from each other within a single tour request. The values $1, \frac{1}{2}, \frac{1}{3}$ and $\frac{1}{5}$ were tested so that the obtained tours differ in 100%, 50%, 33% or 20% of visit points.

Several tour requests were executed on different maps. Results show that the most usable values are $\frac{1}{2}$ and $\frac{1}{3}$. These values provide the perfect pool of tours so that the user can benefit from the other variations of the best tour. See the differences in these similarity constants in figures A.1 and A.2. For other experiments the $R = \frac{1}{3}$ was used.

For R = 1 tours must be different in all visit points so this value is a very hard constraint which can lead to not finding the K tours as requested. On the

other hand $R = \frac{1}{5}$ makes the difference in tours so small that it does not provide enough options of trips for the user. But this value can be useful if we are looking for very little variations of the best tour.

7.2.3 Experiments with improvements per time

In these experiments, I analyze the progress of profit improvements with respect to the calculation time. Graphs 7.2, 7.3, 7.4 show profit improvements of best tours per calculation time for sample tour requests on three different maps. Each line in these graphs is a single run of the Tour Searching algorithm for the maximum calculation time of 10 minutes. The logarithmic scale is used for the calculation time since the majority of improvements occurs within a first few seconds. Sample tour requests with different time budgets were repeated 10 times to see how these runs differ.

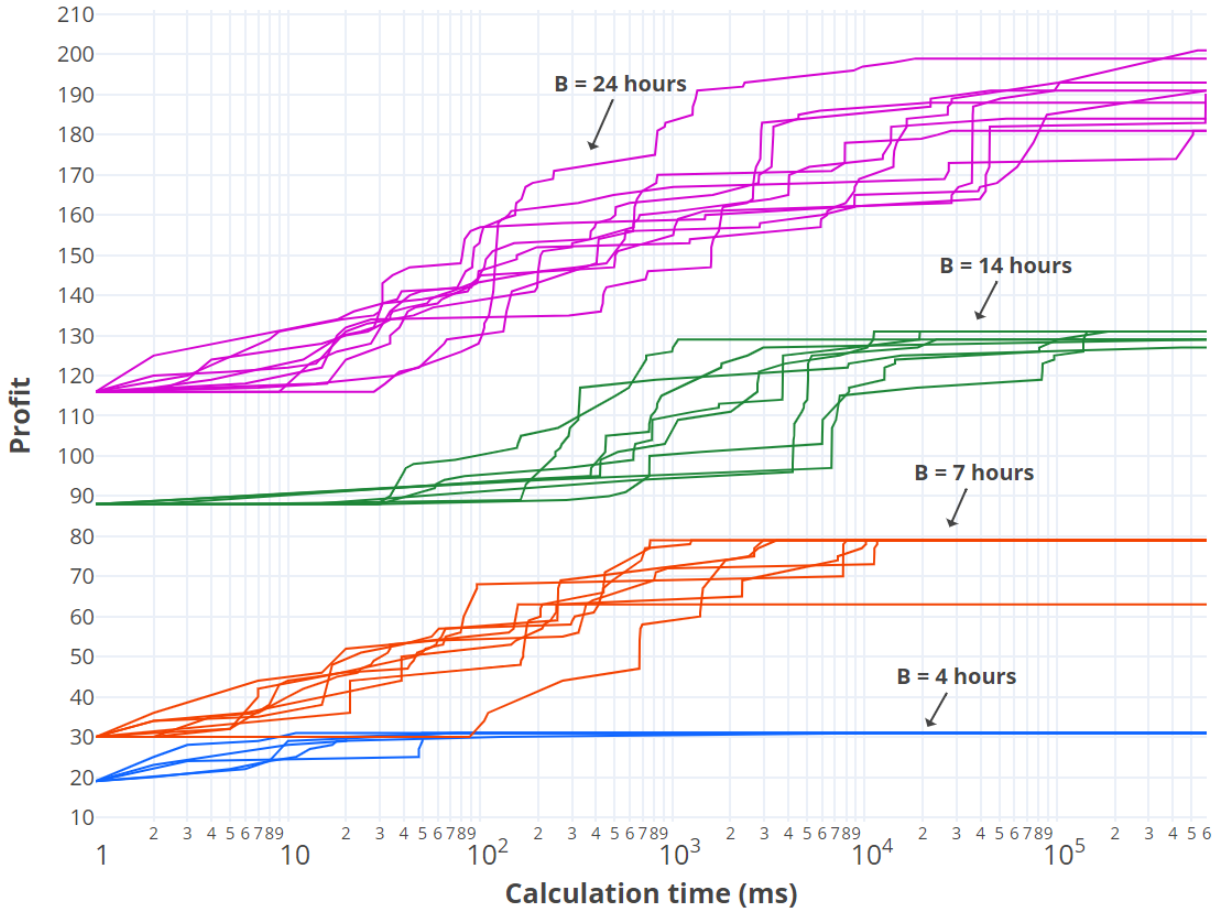


Figure 7.2: Profit improvements with respect to the calculation time for different time budgets (B), measured on the Bohemian Forest map.

The results show that the Tour Searching algorithm achieves the greatest profit improvements within the first second of the execution (10^3 ms). For all maps, the maximum profit was reached within 2 seconds for time budgets up to 7 hours. With the increasing time budget, a longer calculation time was needed to converge to the profit maximum and the occurrence of runs where the Tour Searching algorithm got stuck in a local optimum increases. These suboptimal runs are eliminated by the multithreaded implementation of the ATPS which executes the TS algorithm several times and obtains the best result from these executions. But despite this, for time budgets above 7 hours, the TS algorithm reaches almost maximal profits after 10-20 seconds of execution on maps up to 360 VP. The further extension of the ATPS can set the calculation time based on the estimation from the number of VP.

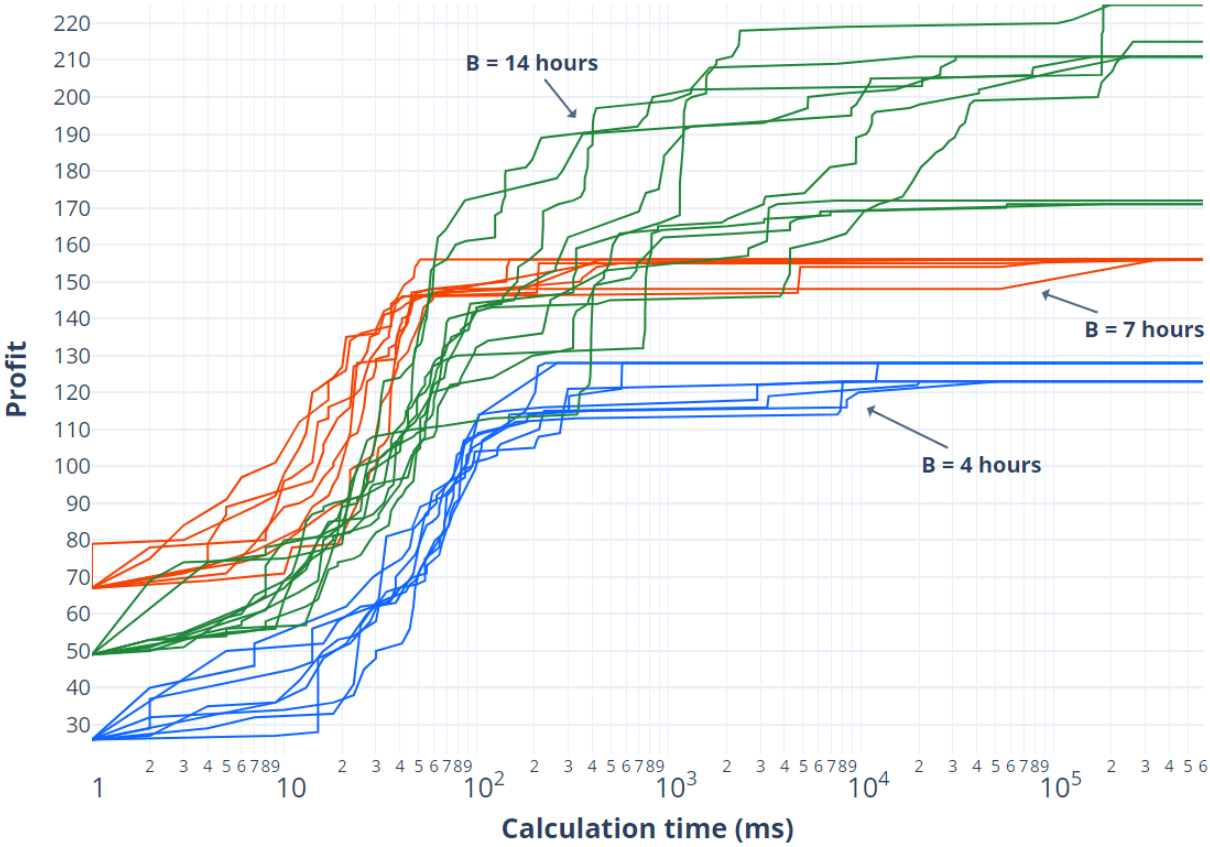


Figure 7.3: Profit improvements with respect to the calculation time for different time budgets (B), measured on the Bohemian Paradise map.

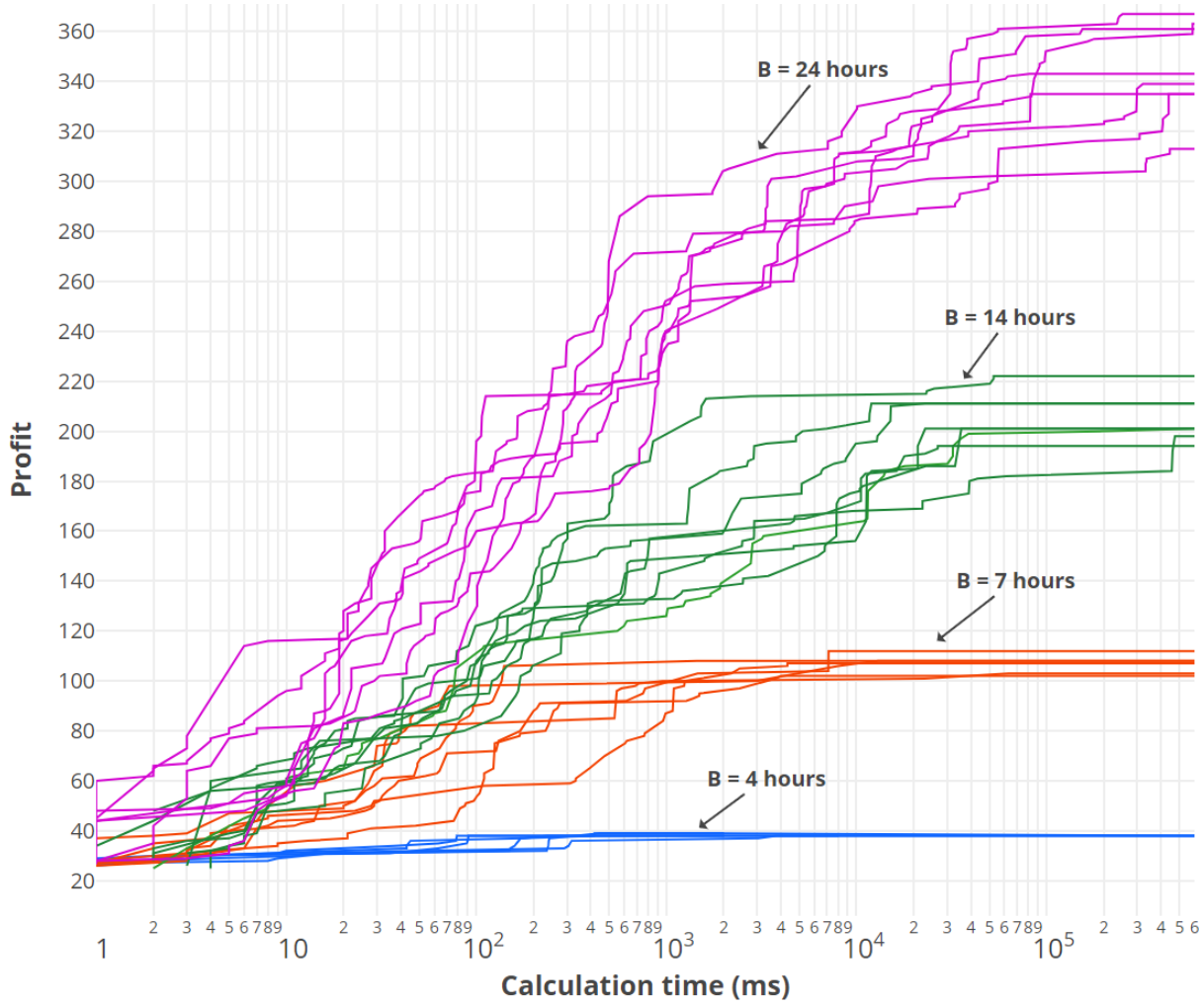


Figure 7.4: Profit improvements with respect to the calculation time for different time budgets (B), measured on the Giant Mountains map.

7.2.4 Experiments with determinization

In these experiments, I measure the average profit, the standard deviation (σ) and the relative standard deviation (CV) of the profit value for three sample tour requests on three different maps. The goal of this experiment is to find out how much the results from the ATPS are deterministic and how these results deviate with respect to the number of visit points and also with respect to the input time budget B.

Several sample tour requests were performed on three different maps (Jesenik, Bohemian Forest, Giant Mountains). The start and the goal positions

were fixed for each map. Time budget values were 4, 7, 14, 21, 28 hours which simulates 1 to 4 day trips. Each tour request was performed ten times to gather average profit values and deviations. This whole testing was done for two calculation times of the ATPS, 2 and 5 second.

Three tables below show the results obtained from this testing. For each of the tour requests, the average profit of the best 5 tours found by the ATPS was calculated and then averaged over ten runs of the system. It means that the average profit value ($P_{average}$) in these tables illustrates the overall profit valuability of the best 5 tours found by the ATPS. The standard deviation (σ) shows how these results differ in the profit value and the relative standard deviation (CV) shows how these results deviate in relation to the average profit.

The results proved several hypothesis. All average profits on all three maps increased with the higher calculation time of the ATPS. The difference is greater on larger maps with wider search space of possible tours, especially the Giant Mountains map. The σ and CV are also higher for larger maps and for the time budget value for which the number of possible tours is the largest.

Calc. Time:	**2 seconds**			**5 seconds**		
B **(hours)**	$P_{average}$	σ	CV	$P_{average}$	σ	CV
4	64,0	0,0	0,0 %	64,0	0,0	0,0 %
7	98,8	3,2	3,2 %	99,8	1,8	1,8 %
14	133,6	4,2	5,6 %	139,2	3,0	4,1 %
21	170,0	5,3	9,0 %	179,2	4,9	8,8 %
28	217,8	8,2	17,9 %	241,0	4,3	10,4 %

Table 7.4: Results obtained from 10 executions of a sample tour request on the Jesenik map.

Calc. Time:	**2 seconds**			**5 seconds**		
B **(hours)**	$P_{average}$	σ	CV	$P_{average}$	σ	CV
4	31,8	2,0	0,6 %	32,0	0,0	0,0 %
7	61,0	2,3	1,4 %	61,2	1,7	1,0 %
14	101,6	8,6	8,7 %	106,0	7,0	7,4 %
21	150,0	13,4	20,1 %	152,2	7,1	10,8 %
28	200,8	5,0	10,1 %	210,8	4,4	9,4 %

Table 7.5: Results obtained from 10 executions of a sample tour request on the Bohemian Forest map.

Calc. Time:	**2 seconds**			**5 seconds**		
B **(hours)**	$P_{average}$	σ	CV	$P_{average}$	σ	CV
4	67,4	1,4	1,0 %	67,6	1,2	0,8 %
7	151,7	2,5	3,8 %	155,7	2,0	3,1 %
14	212,7	6,5	13,7 %	230,2	4,3	9,9 %
21	264,3	8,6	22,8 %	279,6	9,6	26,8 %
28	325,7	7,4	24,3 %	339,5	10,5	35,8 %

Table 7.6: Results obtained from 10 executions of a sample tour request on the Giant Mountains map.

For all maps, the CV of average profits with time budgets up to 7 hours was less than 3,8%. It means that the ATPS provides almost deterministic results in the sense of obtained profit for one-day trips.

The CV for the Jesenik map without SOI was less than 10 % even for the 21 hour time budget. It means that the repeated calculations of the ATPS for this request will differ at most by 10 % of the average profit value from 5 best tours found. Almost the same results were obtained for the Bohemian Forest map with the difference that 5-second calculation time was needed to get results with difference of 10 % CV. These two results show that the ATPS provides very similar results in the sense of obtained profit on maps up to 150 VP, with the time budget up to 21 hours.

The differences between the Jesenik map and the Giant Mountains map shows how an integration of the SOI affects this task. Segments make the search space much wider so the longer calculation time is needed to obtain better results. The results from the Giant Mountains map show that for the 360 VP the average profit value of the tours obtained differ in less than 14 % for the time budget up to 14 hours. Higher time budgets make the task complexity on this map much harder and the obtained results differ more. On the other hand, the probable real use cases are more similar to smaller map areas, so this map can be divided into smaller locations according to the current position of the user. Then the results obtained will be similar to the results for smaller maps.

7.2.5 Real use case scenarios

In this section, I perform experiments which are as similar as possible to the real use case scenarios. I have created several tour requests which correspond to the real trip requirements in well-known locations. The ATPS provided results for these trips within 2 seconds. I have evaluated the obtained results with the emphasis on the trip quality in these locations, mainly if the trip fulfills basic expectations. Some of these results are presented in this thesis. All results are available in the attachment.

Round trips

The goal of this testing is to find out if the ATPS finds pleasant round trips in the selected locations. Frequently required tour requests were created for the Bohemian Paradise location. The preview of an example round trip shows the figure 7.5. See additional tour results in the figures A.3 and A.4.

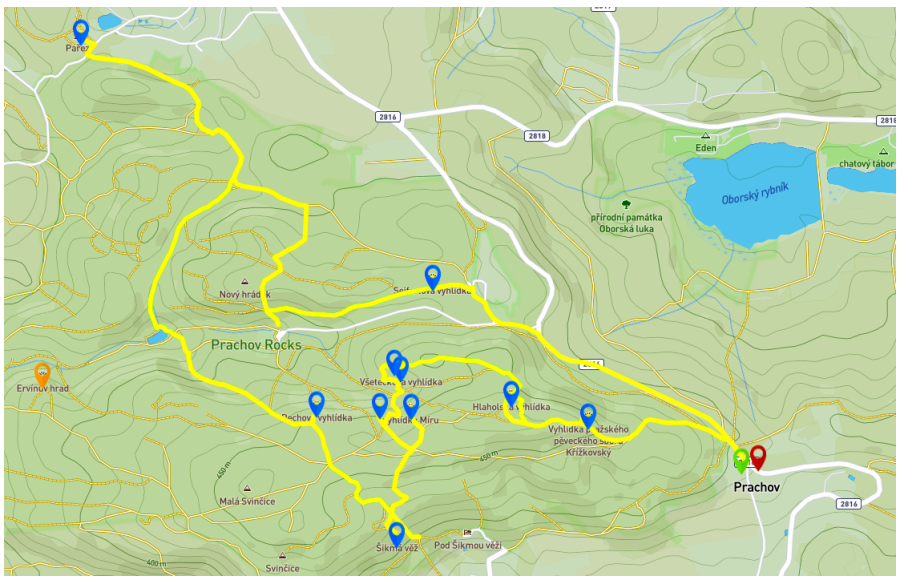


Figure 7.5: An example 4-hour round trip in the Bohemian Paradise from Prachov. Found by the ATPS in 2 seconds.

Long hiking trips

The goal of this testing is to find out if the ATPS finds pleasant long hiking trips, e.g. 2-day trips over the mountains. Frequently required tour requests were created for the Giant Mountains and the Bohemian Forest locations. The preview of an example hiking trip shows the figure 7.6. See additional tour results in the figures A.5 and A.6.

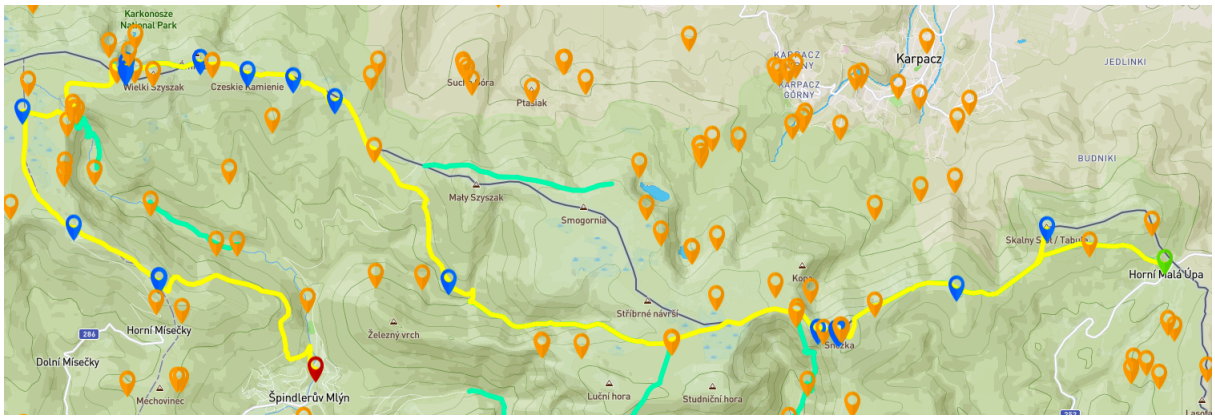


Figure 7.6: An example 14-hour (2-day) hiking trip in the Giant Mountains from Špindlerův Mlýn to Horní Malá Úpa. Found by the ATPS in 2 seconds.

Use case conclusion

The results show that the ATPS finds efficiently trips which are comparable to the trips created by humans. At first sight, the trips visit as many POI as possible and lead along as many SOI as possible. Found tours are efficiently adjusted to Poi profit updates using the recalculation mechanism. There is a minimum of so called return paths which force travelers to go along these paths twice. Even though this evaluation element is not captured in the task, the profit optimization criterion successfully manages to eliminate these return paths. Reviewed trips proved that the ATPS is able to provide a variety of pleasant tourist trips in a very short amount of time.

Chapter 8

Conclusion

In this thesis, I have covered several topics and concepts related to the trip planning in order to create a functional, efficient and applicable trip recommendation system. First of all, I have analyzed existing approaches and solutions which solve some type of trip planning tasks. I have focused on the orienteering problem (OP) [4] and its solvers since this problem models well the basic trip planning task. I have deduced and defined several trip planning components such as map, map with duration function, points of interest, segments of interest and then I have defined my trip planning task with POI and SOI which is an extension of the basic trip planning task modelled as the classic orienteering problem. I have designed the Tour Searching algorithm based on an evolution approach and the VNS algorithm [24]. I have designed the Automated Trip Planning System called ATPS using this algorithm which solves the defined task. The system was designed to be usable on any general map data input and to be easily extended for a specific real-life problem. I have designed the optimal ILP algorithm for the defined task which can be incorporated into the ATPS. The testing data were gathered using the OSM database [31] which turned out to be a good data source for the trip planning problem. The whole system was implemented as the NodeJS server [35] with the C++ Addon containing the main trip planning logic. The frontend client webpage was imeplemented in the javascript using the MapBox API [34]. The created system is a completely functional prototype demonstrating the main trip planning use cases so it can be incorporated or extended to an extensive trip planning recommendation system. The system was tested in two phases. A new test benchmark set was created for this task to evaluate the designed Tour Searching algorithm compared to the optimal ILP algorithm. In the second phase, the system was tested thoroughly using the real-world map data on several locations to adjust all parameters and to present results of several real use cases.

The results show that this system creates good-quality multiple trip recommendations in a few seconds. The system is able to provide a set of 5 distinct tours of a satisfactory quality in 2 seconds for any tour request on a medium-sized map. It can also be set up to focus on the best tour only if we require only a single result with the most profit value. Comparison with the optimal ILP algorithm shown that the Tour Searching algorithm reaches almost optimal profit values within a few seconds and it hardly overcomes the optimal algorithm in execution times for large instances of the task. With an increased calculation time, it can be used on many different real-life optimization problems not related to the trip planning.

The evolution approach proved to be a very effective technique in the recalculation process which allows the user to adjust some parameters and request new results in an even shorter calculation time. Using this technique, the system can be easily extended with many user interactive elements to create trips for the actual user's preferences indeed.

For future work, an extension of the individual to represent time windows or time dependencies might be added. The individual concept also allows to implement easily some hard constraints (must visit, not allowed to visit). For an easy deployment of this system, the map preprocessing and loading mechanisms might be improved so that it is possible to use the ATPS effectively on the global map input.

Bibliography

[1] W. Souffriau and P. Vansteenwegen, "Tourist trip planning functionalities: State–of–the–art and future," in *Current Trends in Web Engineering* (F. Daniel and F. M. Facca, eds.), (Berlin, Heidelberg), pp. 474–485, Springer Berlin Heidelberg, 2010.

[2] X. Lu, C. Wang, J.-M. Yang, Y. Pang, and L. Zhang, "Photo2trip: Generating travel routes from geo-tagged photos for trip planning," in *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, (New York, NY, USA), p. 143–152, Association for Computing Machinery, 2010.

[3] Z. Chen, H. T. Shen, and X. Zhou, "Discovering popular routes from trajectories," in *2011 IEEE 27th International Conference on Data Engineering*, pp. 900–911, April 2011.

[4] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1 – 10, 2011.

[5] A. C. Leifer and M. B. Rosenwein, "Strong linear programming relaxations for the orienteering problem," *European Journal of Operational Research*, vol. 73, no. 3, pp. 517 – 523, 1994.

[6] A. Gionis, T. Lappas, K. Pelechrinis, and E. Terzi, "Customized tour recommendations in urban areas," in *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 313–322, 2014.

[7] M. Fischetti, J. J. S. González, and P. Toth, "Solving the orienteering

problem through branch-and-cut," *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 133–148, 1998.

[8] R. Ramesh, Y.-S. Yoon, and M. H. Karwan, "An optimal algorithm for the orienteering tour problem," *ORSA Journal on Computing*, vol. 4, no. 2, pp. 155–165, 1992.

[9] D. Cavada, N. Mirzadeh, F. Ricci, and A. Venturini, "Interactive itinerary planning with trip@ dvice," *Human-Computer Interaction INTERACT'03 M. Rauterberg et al.(Eds.) IFIP*, pp. 1105–1106, 2003.

[10] V. Soo and S.-H. Liang, "Recommending a trip plan by negotiation with a software travel agent," vol. 2182, pp. 32–37, 09 2001.

[11] L. Ardissono, A. Goy, M. Segnan, and P. Torasso, "Ubiquitous user assistance in a tourist information server," pp. 14–23, 05 2002.

[12] K. J. Cha, "Experimental evaluation of an expert system for travel recommender systems," *International journal of software engineering and its applications*, vol. 8, pp. 115–128, 2014.

[13] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden, "The city trip planner: An expert system for tourists," *Expert Systems with Applications*, vol. 38, no. 6, pp. 6540 – 6546, 2011.

[14] E. H. Lu, C. Lin, and V. S. Tseng, "Trip-mine: An efficient trip planning approach with travel time constraints," in *2011 IEEE 12th International Conference on Mobile Data Management*, vol. 1, pp. 152–161, June 2011.

[15] Y. Zheng, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proceedings of International conference on World Wide Web 2009*, April 2009. WWW 2009.

[16] W. Souffriau, P. Vansteenwegen, J. Vertommen, and G. Vanden Berghe, "A personalized tourist trip design algorithm for mobile tourist guides," *Applied Artificial Intelligence*, vol. 22, pp. 964–985, 10 2008.

[17] W. Souffriau and P. Vansteenwegen, "Tourist trip planning functionalities: State–of–the–art and future," in *Current Trends in Web Engineering* (F. Daniel and F. M. Facca, eds.), (Berlin, Heidelberg), pp. 474–485, Springer Berlin Heidelberg, 2010.

[18] T. Tsiligirides, "Heuristic methods applied to orienteering," *Journal of the Operational Research Society*, vol. 35, pp. 797–809, Sep 1984.

[19] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[20] R. Ramesh and K. M. Brown, "An efficient four-phase heuristic for the generalized orienteering problem," *Computers and Operations Research*, vol. 18, no. 2, pp. 151 – 165, 1991.

[21] S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[22] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[23] I.-M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem," *European Journal of Operational Research*, vol. 88, no. 3, pp. 475 – 489, 1996.

[24] Z. Sevkli and F. E. Sevilgen, "Variable neighborhood search for the orienteering problem," in *Computer and Information Sciences – ISCIS 2006* (A. Levi, E. Savaş, H. Yenigün, S. Balcısoy, and Y. Saygın, eds.), (Berlin, Heidelberg), pp. 134–143, Springer Berlin Heidelberg, 2006.

[25] M. Verhoeven, E. Aarts, and P. Swinkels, "A parallel 2-opt algorithm for the traveling salesman problem," *Future Generation Computer Systems*, vol. 11, no. 2, pp. 175 – 182, 1995. Massive Parallel Computing.

[26] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis, "Heuristics for the time dependent team orienteering problem: Application to tourist route planning," *Computers and Operations Research*, vol. 62, pp. 36 – 50, 2015.

[27] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, vol. 7, no. 1, pp. 65 – 85, 1988.

[28] T. Bektaş and L. Gouveia, "Requiem for the miller–tucker–zemlin subtour elimination constraints?," *European Journal of Operational Research*, vol. 236, no. 3, pp. 820 – 832, 2014. Vehicle Routing and Distribution Logistics.

[29] W. G. G. H. K. Küstner, *The VNR Concise Encyclopedia of Mathematics*. Springer, Dordrecht, 1990.

[30] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193 – 2196, 2012.

[31] O. Foundation, "Open street map." [Online] Available: `https://www.openstreetmap.org`. Accessed 2019-11-20.

[32] O. Foundation, "Overpass api." [Online]. Available: `https://wiki.openstreetmap.org/wiki/Overpass_API`. Accessed 2019-11-20.

[33] O. Foundation, "Osmosis java tool." [Software]. Available: `https://wiki.openstreetmap.org/wiki/Osmosis`. Accessed 2019-12-10.

[34] MapBox, "Mapbox api." [Software]. Available: `https://mapbox.com`. Accessed 2019-12-10.

[35] NodeJS, "Nodejs." [Software]. Available: https://nodejs.org. Accessed 2019-12-10.

[36] L. Gurobi Optimization, "Gurobi optimization." [Software]. Available: https://www.gurobi.com. Accessed 2020-02-02.

[37] Seznam.cz, "Mapy.cz." [Online] Available: https://www.mapy.cz. Accessed 2019-11-20.

[38] T. LLC, "Tripadvisor." [Online] Available: https://www.tripadvisor.cz. Accessed 2019-11-20.

[39] S. Costa, R. Coles, and A. Boltwood, "Landscape experience and the speed of walking," 09 2015.

Appendix A

Gallery of found tours

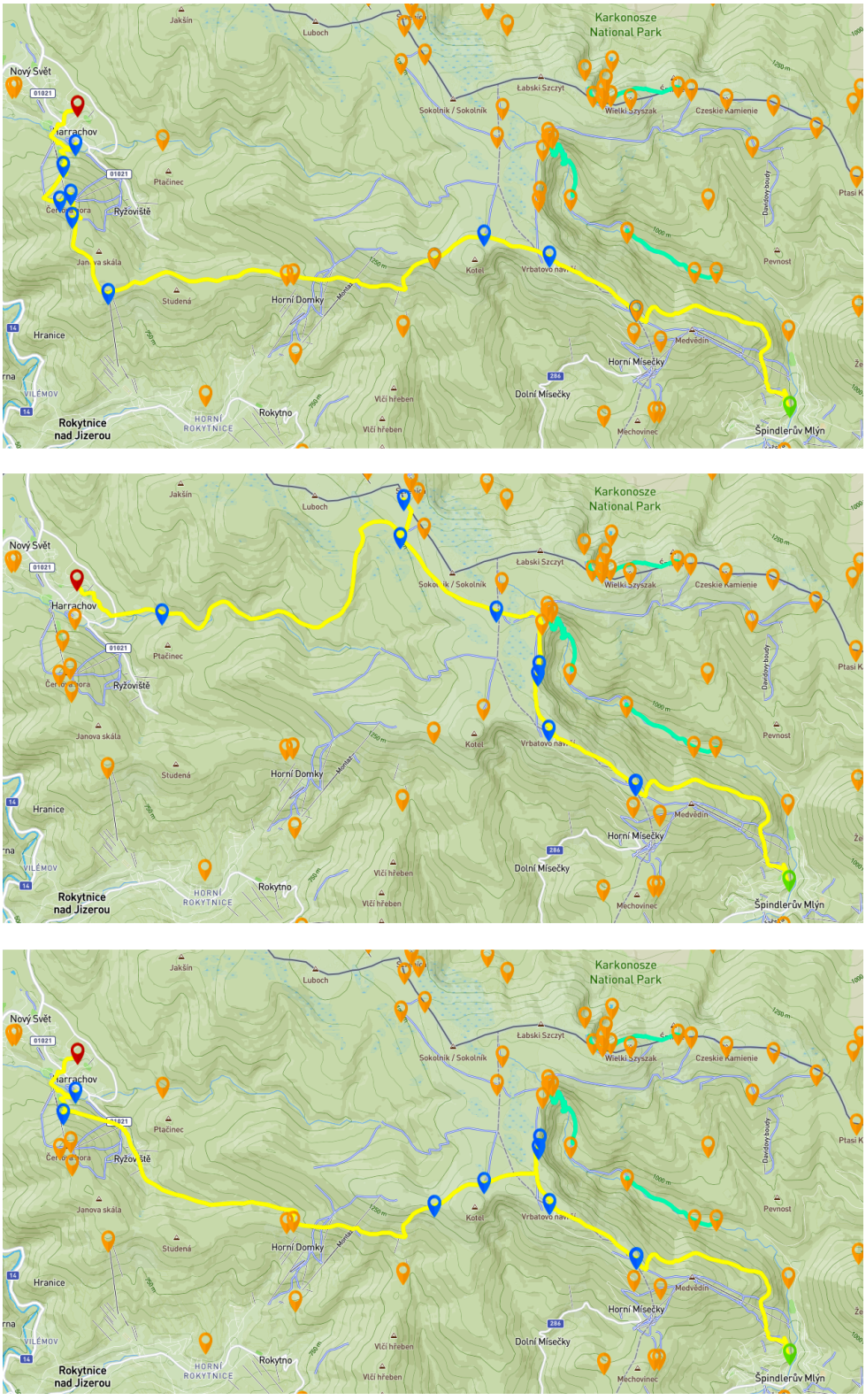


Figure A.1: The best 3 tours found for a 7-hour trip in the Giant Mountains from Špindlerův Mlýn to Harrachov using the similarity constant R = 1/2.

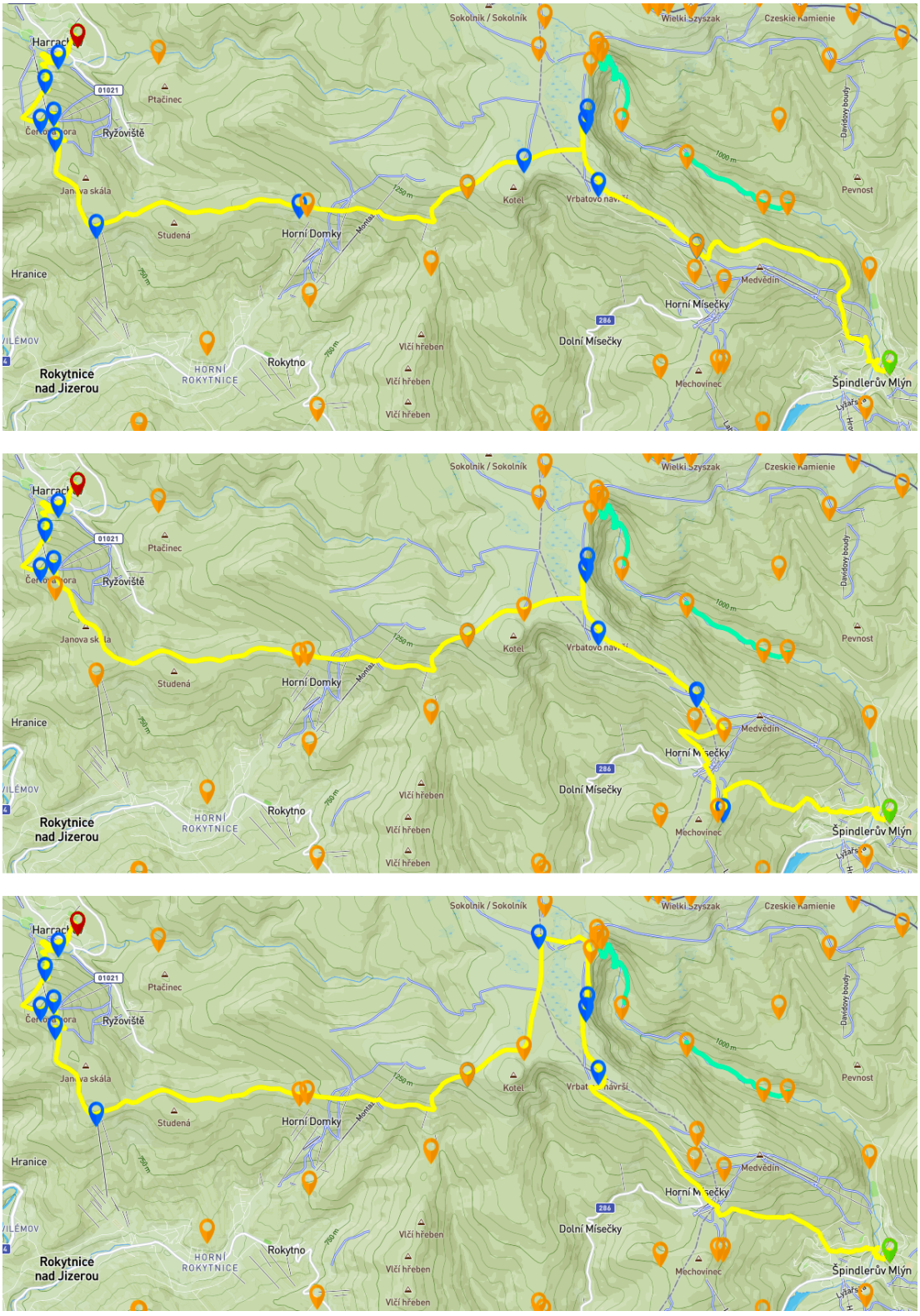


Figure A.2: The best 3 tours found for a 7-hour trip in the Giant Mountains from Špindlerův Mlýn to Harrachov using the similarity constant R = 1/3.

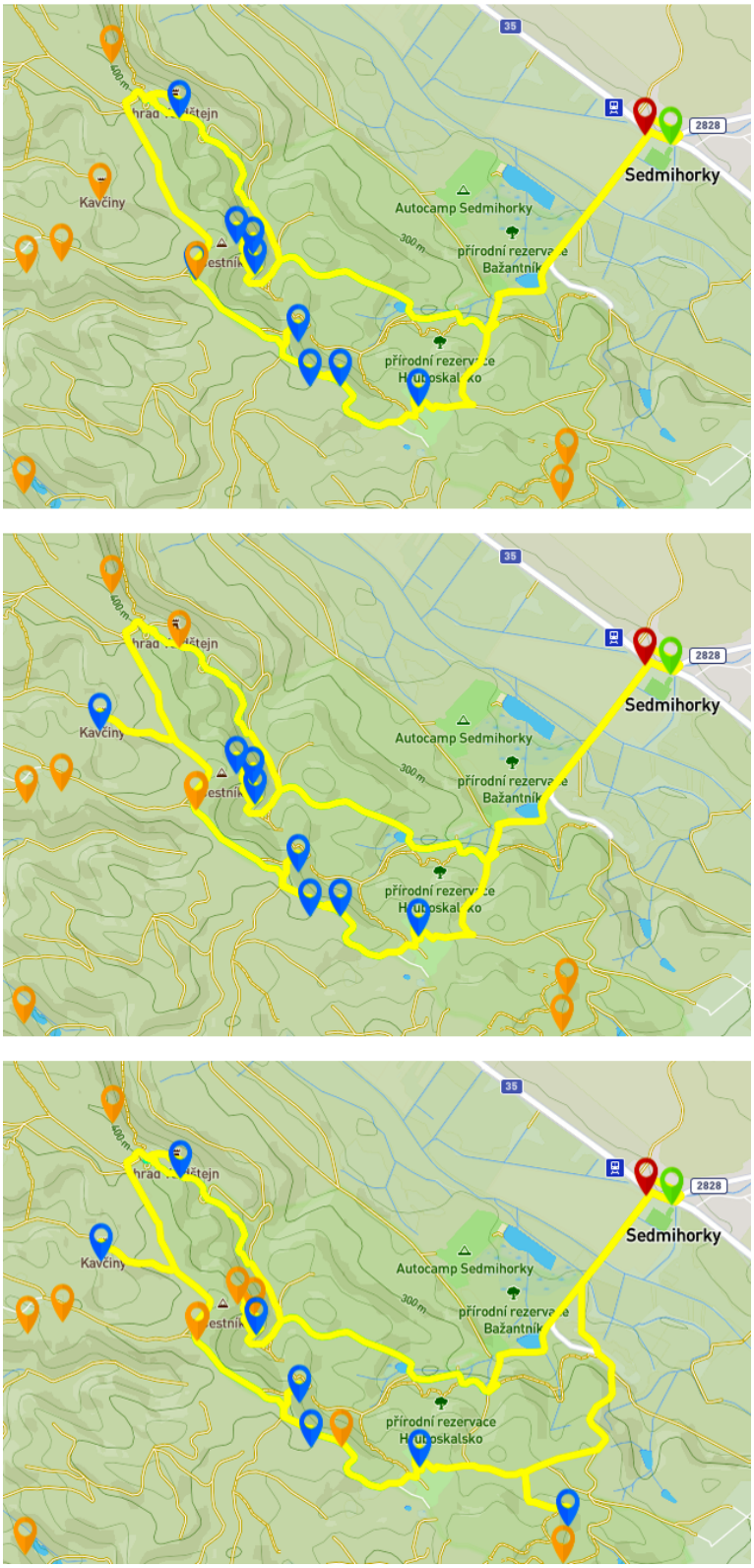


Figure A.3: The best 3 tours found for a 4-hour round trip in the Bohemian Paradise from Sedmihorky. Found by the ATPS in 2 seconds.

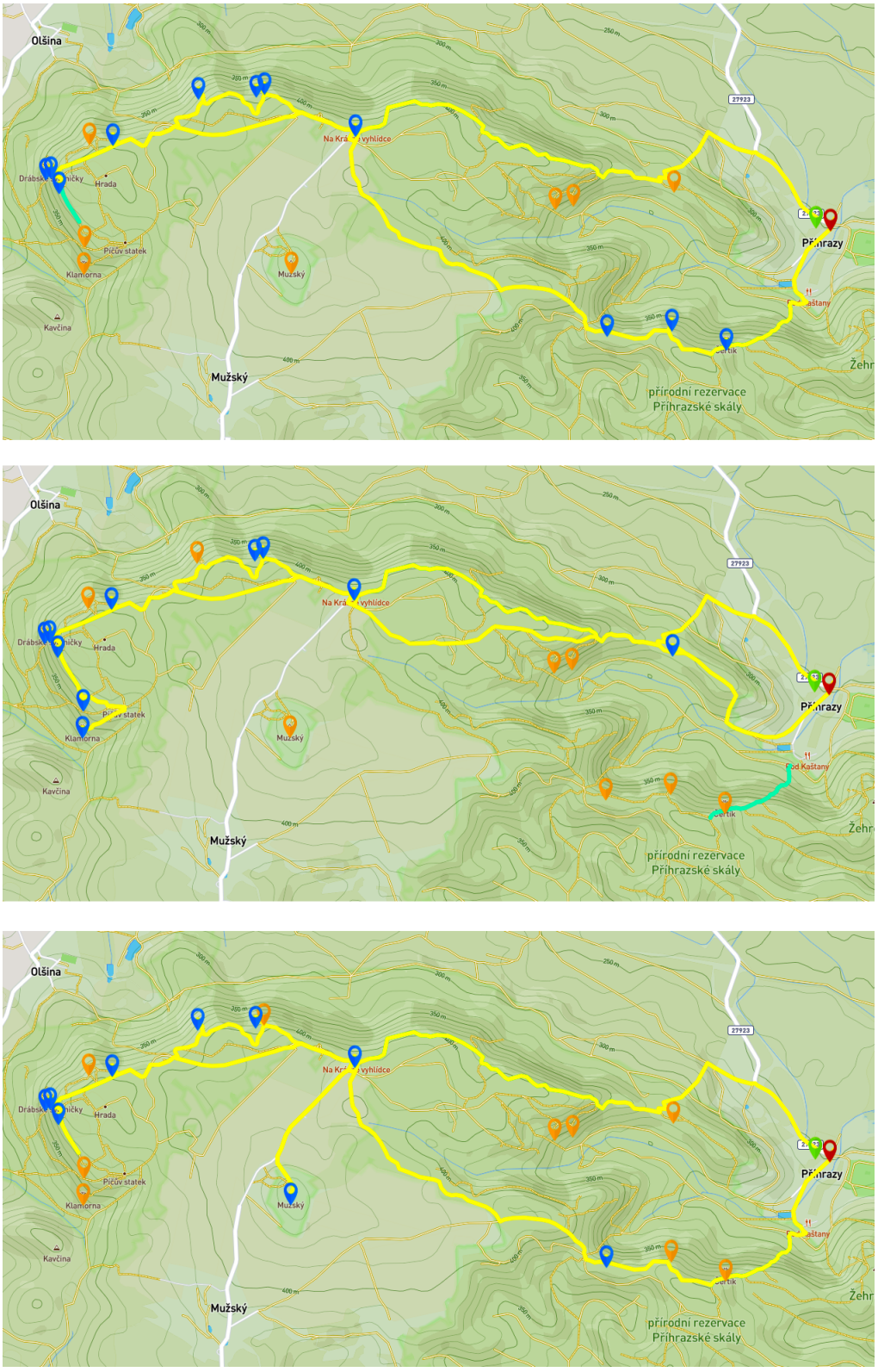


Figure A.4: The best 3 tours found for a 4-hour round trip in the Bohemian Paradise from Příhrazy. Found by the ATPS in 2 seconds.

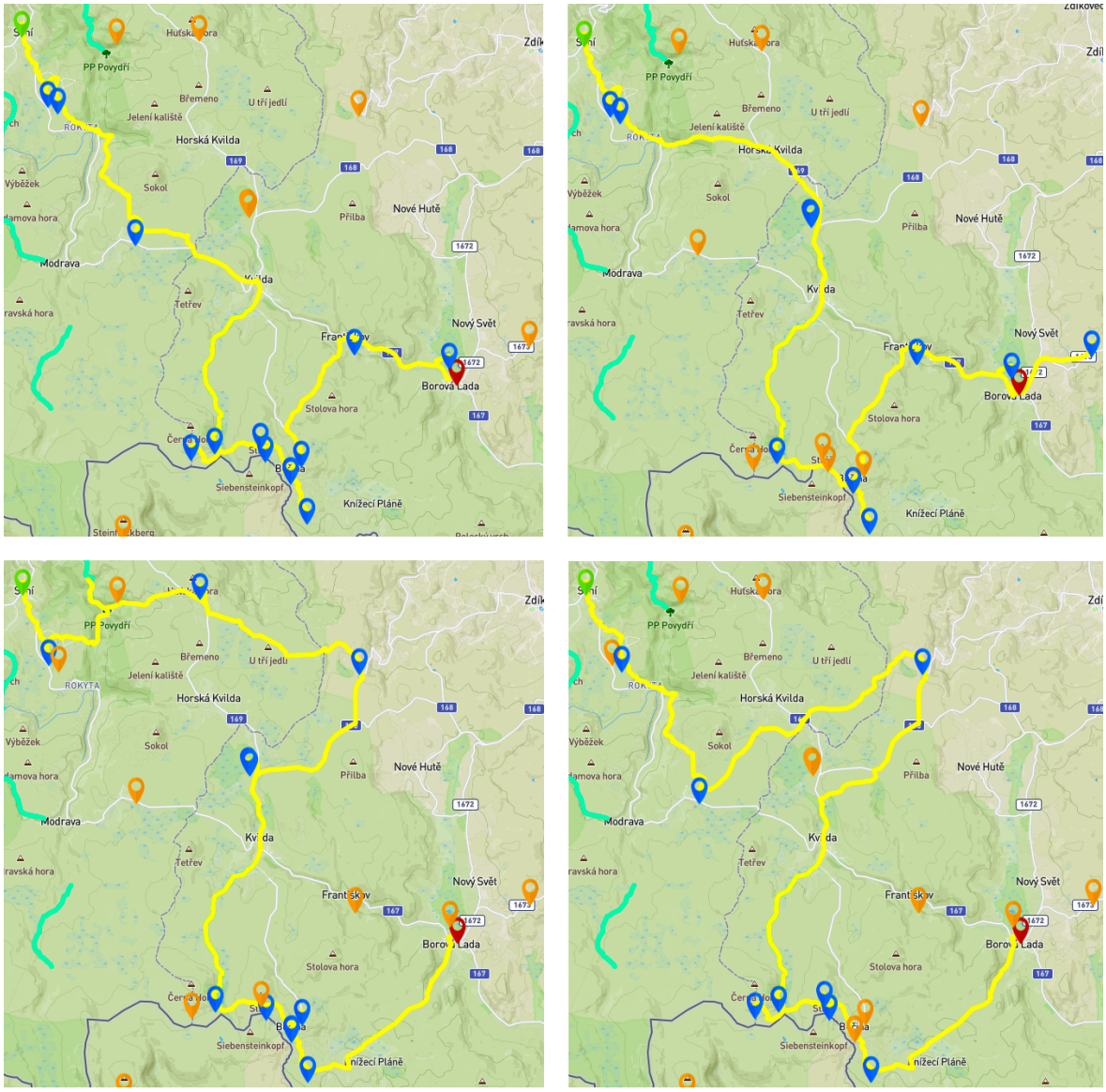


Figure A.5: The best 4 tours found for a 16-hour tour request, a 2-day hiking trip, in the Bohemian Forest. Found by the ATPS in 2 seconds.

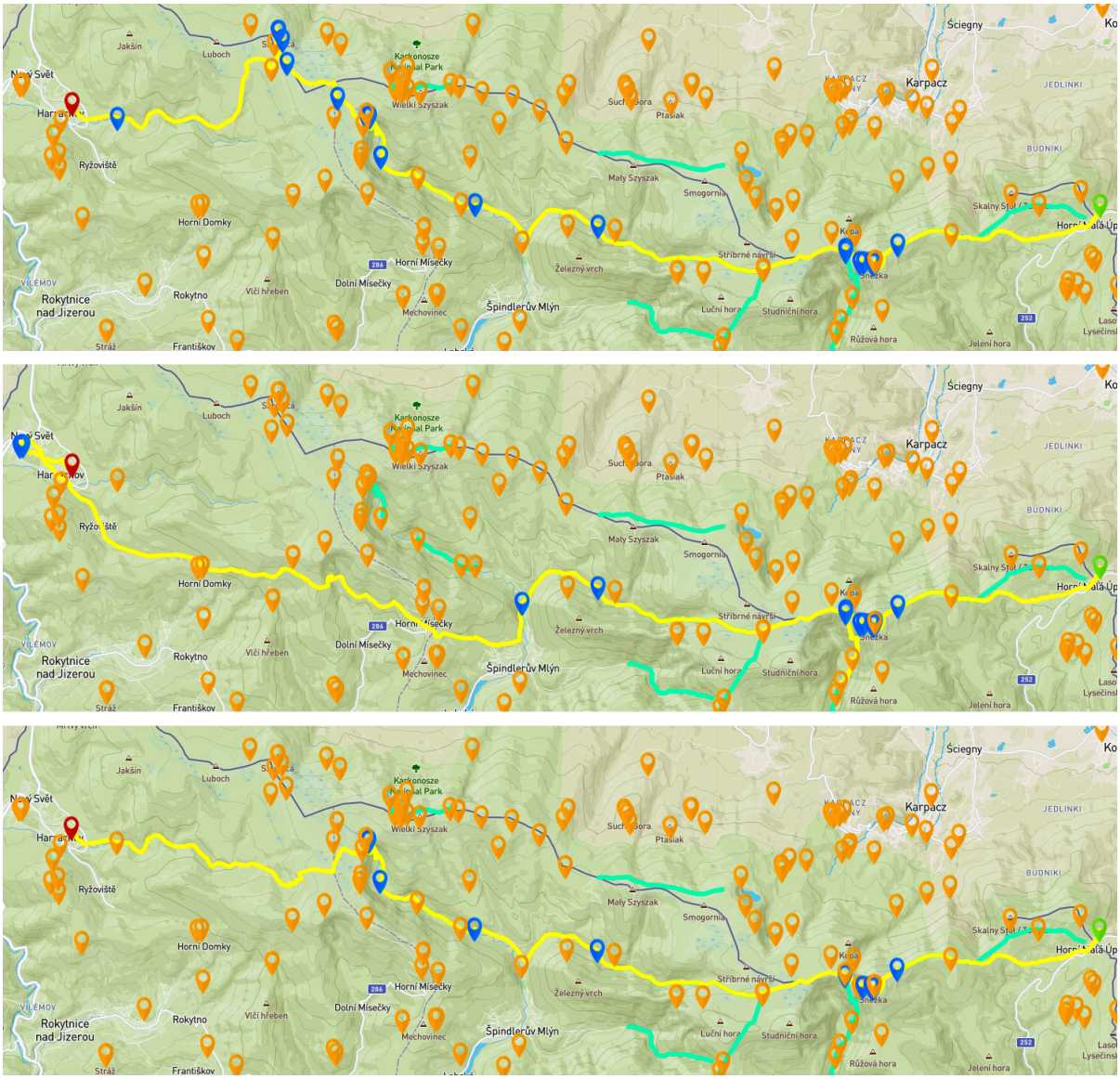


Figure A.6: The best 3 tours found for a 16-hour tour request, a 2-day hiking trip, in the Giant Mountains. Found by the ATPS in 2 seconds.

Appendix B

Abbreviations

ATPS Automated Trip Planning System

TTDP Tourist Trip Design Problem

OP Orienteering Problem

VNS Variable Neighborhood Search

ILP Integer Linear Programming

POI Point Of Interest

SOI Segment Of Interest

SDP Segment Division Point

VP Visit Point

DM Distance Matrix

PM Profit Matrix

TS Tour Searching

API Application Programming Interface

XML Extensible Markup Language

OSM Open Street Map

GPS Global Positioning System

CV Coefficient of Variation

Appendix C

Contents of the enclosed CD

- `ATPS` ..... Automated Trip Planning System implementation
  - `DoxyGen` ..... documentation
  - `maps` ..... map database
  - `test_instances` ..... test benchmark instances
  - `TripPlanner` ..... source code
  - `README.txt` ..... setup manual
- `BasicILP` ..... Basic ILP algorithm implementation
- `results` ..... logs and spreadsheets with measured results
- `thesis` ..... diploma thesis source code in LaTeX
- `tours_gallery` ..... gallery of found tours
- `CDcontent.txt` ..... description of the CD content