# SIGHTSEEING ROUTE PLANNING

*Hristijan Gjoreski, Božidara Cvetković, Boštjan Kaluža, Mitja Luštrek*
Department of Intelligent Systems, Jožef Stefan Institute,
Jožef Stefan International Postgraduate School,
e-mail: {hristijan.gjoreski, boza.cvetkovic, bostjan.kaluza, mitja.lustrek}@ijs.si

**ABSTRACT**

**Route planning is a challenging task because it is a combination of theoretically well-defined computational problems on one side, and everyday-life decisions and constraints on the other side. This paper presents an approach to sightseeing route planning using theory of computation. In particular, in this paper we discuss the combination of two well-known computational problems: knapsack and travelling salesman, and their practical implementation in everyday life task – route planning. The algorithms are adapted in such way that they find near optimal solution with minimum delay, almost in real-time. The final result of the algorithms is a suggested list of tourist attractions ordered by their location and attractiveness.**

## 1 INTRODUCTION

Tourism is very important branch for the economy of a country. Its impact is not only economic, it also promotes a country abroad and raises the awareness of our cultural and natural heritage at home. In order to improve the tourism branch in a country, tourists need information on the places of visit delivered in an efficient and attractive fashion. This is often a difficult task, since such information is scattered across various publications and websites. One of the main objectives of the e-Turist project [3] is collecting all the useful touristic information in one place and extracting useful touristic information in order to help the tourists to plan their trip. This information is presented using web and smartphone applications and mainly consists of recommendations of tourist attractions and suggestions of near optimal routes in order to visit all the selected tourist attractions in the time frame available.

This paper presents an approach to sightseeing route planning, which is a key component in the e-Turist project. Route planning in general is a challenging task because it is a combination of theoretically well-defined computational problems on one side, and everyday-life decisions and constraints on the other side. In particular, in this paper we discuss two well-known problems: knapsack problem and travelling salesman problem, and their practical implementation in everyday life task – route planning.

## 2 PROBLEM DESCRIPTION

The problem discussed in this paper addresses the question: "How to plan your sightseeing route, once you have a list of tourist attractions?". In other words, our solution tries to find the near optimal sightseeing route, given the points of interest and the time available for sightseeing.

To explain the basic concept, let us consider the following example, shown in Figure 1. Each of the small boxes represents a point of interest (POI). Additionally, each POI has two features: a fixed visit time duration (the average time a tourist spends at the POI) marked with $W_i$, and an POI evaluation mark (a number from 1 to 5), representing the attractiveness of the POI) marked with $V_i$. On the other hand, there is a tourist, who is limited by time, $W_{max}$, and has only 7 hours for sightseeing. The problem here is how to find the best route (combination of POIs) given the tourist's time limit and the list of the POIs.

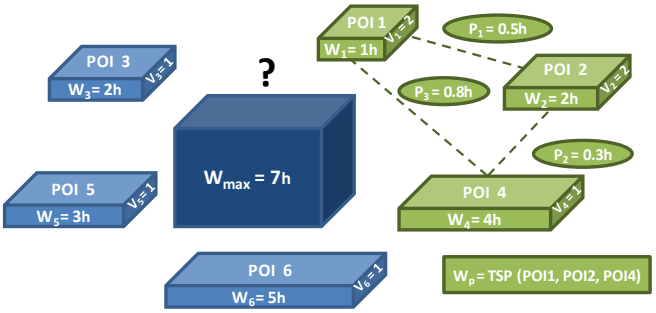


*Figure 1: Modified knapsack problem.*

Current description of the problem reminds of the known knapsack problem [5], with the following parameters: weight – POI visit duration ($W_i$), and a value – POI evaluation ($V_i$). The tourist's time limit is the maximum weight that the knapsack can hold ($W_{max}$). The optimal solution for the example shown in Figure 1 is marked with green color; POI number: 1, 2 and 4. It has a total value of 5 ($V_1 + V_2 + V_4$) and total weight of 7 hours ($W_{total} = W_1 + W_2 + W_4$), which is also the maximum time that the tourist has for his/hers sightseeing route. With this definition, the problem can be solved in a pseudo-polynomial time with dynamic programming [2]. However, this definition does not include the path duration (duration needed to visit all the POIs). In the example shown in Figure 1, that is the path duration to visit POI 1, POI 2 and POI 4. If we assume that

the path duration is symmetric (the same duration stands for POI1→POI2 and PO2→POI1), there are three different path combinations:

- POI1→POI2→POI4
- POI1→POI4→POI2
- POI2→POI1→POI4

It is easy to check that the best (minimum duration) path is POI1→POI2→POI4, which lasts for 0.7 hours ($W_p = P_1 + P_2$). If this path duration is added to the previous total time ($W_{total}$) the new total weight is 7.7 hours which is more than the tourist's maximum time of 7 hours ($W_{max}$). Therefore, this solution should be discarded.

The path duration estimation problem opens a new sub-problem inside the knapsack problem, i.e., how to find the path route with minimum duration, given the POIs. This sub-problem is also a known problem in the theory of computation, called travelling salesman problem (TSP) [7]. In the following sections our proposed solution is explained and the final implementation is presented.

## 3 ALGORITHM

As described in the previous section, we try to find a solution to a knapsack problem, where the weight value changes dynamically (with each algorithm iteration) and it depends on the "boxes" (POIs) chosen. Additionally, the path estimation is computationally expensive process, because it requires solving an NP-hard problem, i.e., TSP. Moreover, the final algorithm execution time should be in the range of several seconds, because it will be used in a real-time POI recommendation application, where the user needs instant feedback from the system. Because of these reasons, several simplifications were proposed: greedy approach for knapsack problem (POIs ordered by value), adapted TSP for path duration estimation (finds near optimal solution).

The first step in our algorithm is the estimation of the importance of a POI (how good a POI is – POI value). For this reason we created a special mathematical definition that considers three factors:

(1) POI's evaluation value
(2) POI's visit duration
(3) POI's local reachability duration

The first factor is a value that varies from 1 to 5 and it represents rough estimation of how interesting a POI is, based on several aspects: the attractiveness, sustainability, visit price, etc. The next factor, the POI's visit duration, represents the average time that a tourist needs in order to see the POI. This is also hardcoded by a domain expert. The final factor, the POI's reachability duration, is a variable that represents how far a POI is from its nearest neighbors. In other words, if a POI is far from the rest of the POIs, the value for this variable would be greater compared to the reachability duration of the rest POIs. Using this information the POIs that are "outliers" (far from the rest of the POIs) are "punished". For the estimation of this variable we used partial implementation of the Local Outlier Detection (LOF) algorithm [1]. In particular, we used the local reachability distance (*lrd*) metric in order to estimate how far a POI is from its neighbors. The LOF algorithm and its mathematical definitions are described by Breunig et al. in their paper, which is also provided in the reference list.

The mathematical definition of the POI importance, which includes all the three factors, is given below.

$$V^* = \alpha * V + (1 - \alpha) * (1 - P^*_{\text{norm}}) * V \quad (1)$$

The variable $V$ is the POI's evaluation value, which varies from 1 to 5. The variable $P^*_{\text{norm}}$, represents the normalized value (from 0 to 1) of the $P^*$, which is a sum of the POI's visit duration (*Vd*) and the POI's local reachability distance (*lrd*):

$$P^* = Vd + lrd \quad (2)$$

Because the idea is to "punish" the POIs which visit lasts longer and the ones that are far away, the normalized $P^*$ is subtracted from 1. The bigger the value of $P^*_{\text{norm}}$ is, the less important the POI is. The $\alpha$ is a parameter regulating the importance of the evaluation value ($V$) on one side, and the POI's visit duration and POI's local reachability distance ($P^*$) on the other side. The empirical analysis of the data showed that 0.5 is a reasonable tradeoff value for $\alpha$. This way, both sides of the equation are equally weighted in the final importance value – $V^*$.

To summarize, a part of the value V (the fraction alpha) is considered as it is, while the rest (the fraction 1 - alpha) is reduced by the factor corresponding to the time needed for the visit (1 - $P^*_{\text{norm}}$).

In the next step of the algorithm, all the POIs are ordered by the importance value, i.e., $V^*$. Next, using a greedy strategy, the algorithm adds items (POIs) in the knapsack until the limit is reached. With each POI added, the weight of the knapsack is checked – if the weight (time duration) of the chosen POI combination is below the maximum weight (total available time of the tourist). In addition with each added POI, a TSP algorithm estimates the path duration, which is also checked with the time limit of the tourist. This way, a near optimal combination of POIs is found.

Once the combination of POIs is found, in the next step it is checked if the user prefers to start from the nearest POI. If this is the case, the order of the POIs is recalculated with a modified version of the original TSP which creates a path using a fixed start POI.

In the final step of the algorithm, it is checked if the tourist has chosen multiple days for sightseeing. In the case of multiple-days visit, the POIs are segmented into groups, each group corresponding to one day of the trip. Additionally, it is checked if the user plans a meal in a particular hour of the day. If this is the case and there are restaurant-POIs in the list of POIs, the best (according to the evaluation value) restaurant is chosen. That day's route

is modified in such a way that the tourist is near the restaurant during the previously chosen meal-time.

### 3.1 Travelling Salesman Problem

In this section the TSP solution algorithm is discussed. As mentioned earlier, the TSP solves a sub-problem in the general knapsack problem. Therefore, its execution time needed to be in the range of several milliseconds. Because the TSP is a NP-hard problem, finding an optimal solution sometimes may be very difficult and computationally expensive. Therefore, for its implementation, we considered an open source algorithm [8], which finds a near optimal solution. It is a greedy approach with additional optimization mechanism. The empirical tests showed that for our scenario (up to 200 POIs) it almost always finds the optimal solution, and also the execution time is acceptable i.e., several milliseconds.

The original algorithm implementation does not take into account a start and end POI. It just connects POIs until a complete path connecting all the POIs is completed. However, in our implementation in some cases, fixed start and end POIs were needed. For this reason two modified versions of the original algorithm were implemented. The first one considers only a start POI and finds the appropriate path using the start POI constraint. This implementation is used in the case the tourist wants to start the sightseeing in the nearest POI. The nearest POI is calculated using the GPS signal of the tourist's smartphone. The second modified version considers not only the start POI, but also the end POI. This version is used in the case of a selected meal-time. In this case the path is divided into two parts: before and after lunch. In the first part the end POI is fixed, which is the restaurant. In the second part, the start POI is fixed, again the restaurant POI.

### 3.2 Algorithm Implementation Optimizations

Once the algorithms were implemented and the number of POIs increased in the range of a 100, few time-related problems emerged.

The first problem was related to the computation of the distance and duration between each of the POIs. To accurately compute the distance and the duration between the POIs, we used the Google maps distance matrix API [5]. Because the API calls are limited, we decided to save the distances into a database. An update of the distances is triggered only when new POI is entered into the database. However, the problem with too many database calls still existed. For each TSP execution, a new distance matrix was computed and therefore too many database calls were executed. That means if 20 POIs are analyzed, then the matrix has size of 20 x 20, which results in 400 database calls. To speed up this process, we decide to save a matrix that contains all the POIs distances into the RAM memory. This way, every time a distance is required, the result is taken out of RAM instead of calling a database. This solution, significantly decreased the time execution.

The second problem that appeared was related to the calls to the TSP algorithm. As mentioned earlier, the TSP problem is NP-hard and thus its execution is computationally expensive process. Therefore, we had to limit the calls to this algorithm. In our first implementation, the TSP was called every time a new POI was added to the solution list. This resulted into too many calls, especially when the number of POIs increased up to 100. Therefore, a solution that limits the calls to the TSP algorithm was suggested. This solution, calls the TSP algorithm only when the time needed to visit all the POIs reached a predefined threshold of 80% of the total available time. In other words, when the time needed to visit the POIs reaches 80% of the total available time, the TSP is called to estimate the exact path duration. Otherwise, every time a POI is added, the path is estimated simply by adding the path duration to the nearest POI.

## 4 IMPLEMENTATION

The sightseeing route planning algorithm was implemented in a more general trip planning application – e-Turist [4]. The general idea of this application is to provide a help to tourists which plan to visit Slovenia. The help consists of a POIs recommendation and organization, route planning, etc. An example of a sightseeing route planning is shown in Figure 2. The figure shows a 2-day trip plan with 11 POIs. The POIs are ordered by their location and the order of visiting is marked with consecutive letters from the alphabet. The POIs marked with yellow color are alternative POIs, which were excluded from the final solution because of the user's time-constraints. However, the user can still decide to visit these alternative POIs. Also a time duration estimation is provided for each day. The estimation is based on the POIs visit durations and the path duration time.

## 5 CONCLUSION

The paper presented an approach to sightseeing route planning. Our task was combining theoretically well-defined computational problems on one side, and everyday-life decisions and constraints on the other side. In particular, in this paper we discussed two well-known problems: knapsack and travelling salesman problem, and their practical implementation in everyday life task – route planning. The algorithms were adapted in such way that they find near optimal solution in with minimum delay, almost in real-time. The final result is a list of tourist attractions ordered by their location and attractiveness.

### Acknowledgement

This work was supported by Slovenian ministry of education, science and sport: call for proposals for co-funding of projects developing e-services and mobile applications for public and private non-profit organizations. The authors would like to thank Andrej Tratnik, Vito Janko and Maja Somrak for the help provided in the system implementation and programming part.

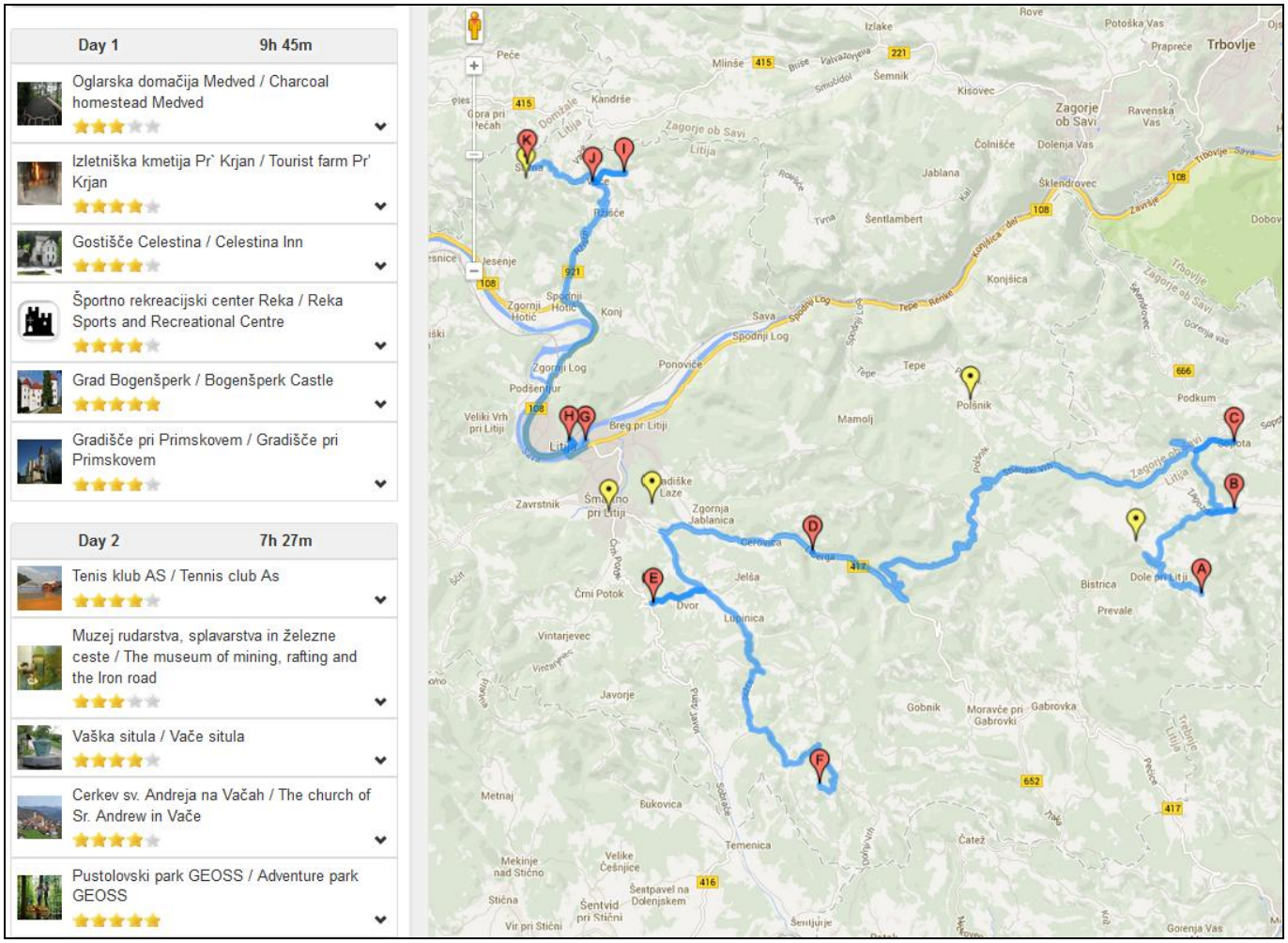


*Figure 2: Example of sightseeing suggested route for two-day trip.*

**References:**


[1] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J. (2000). "LOF: Identifying Density-based Local Outliers". Proceedings of the 2000 ACM SIGMOD international conference on Management of data. SIGMOD '00: 93–104.

[2] Dynamic Programming Knapsack 0-1 Problem. http://www.geeksforgeeks.org/dynamic-programming-set-10-0-1-knapsack-problem/

[3] e-Turist project information. http://dis.ijs.si/e-turist/

[4] e-Turist web application. http://e-turist.ijs.si/

[5] Google Directions API. https://developers.google.com/maps/documentation/distancematrix/

[6] Knapsack problem http://en.wikipedia.org/wiki/Knapsack_problem

[7] Travelling salesman problem. http://en.wikipedia.org/wiki/Travelling_salesman_problem

[8] Travelling salesman problem, Python implementation: https://github.com/dmishin/tsp-solver