Solving touristic trip planning problem by using taboo search approach

Kadri Sylejmani[1, 2] and Agni Dika[1]

**[1] Department of Computer Engineering, Faculty of Electrical and Computer Engineering, University of Prishtina
Prishtina, 10000, Kosovo**

**[2] Faculty of Informatics, Vienna University of Technology
Vienna, A-1040, Austria**

Abstract

In this paper, we introduce an algorithm that automatically plans a touristic trip by considering some hard and soft constrains. Opening and closing hours of POIs (Points of Interest), trip duration and trip allocated budget represent the hard constraints, while the satisfaction factors of the POIs and travelling distance in the trip are considered as soft constraints. We use the soft constraints to evaluate the generated solution of the algorithm. The algorithm is developed by utilizing the taboo search method as a meta heuristic. The operators of Swap, Insert and Delete are used to explore the search space. The Swap and Insert operator are used in each iteration of the algorithm loop, while the Delete operator is used whenever the algorithm tends to enter in an endless cycle. The algorithm is developed by using Java programming language, while the data repositories are created in the XML format. The algorithm is tested with 40 instances of POIs of the city of Vienna. Various entry parameters of the algorithm are used to test its performance. The results gained are discussed and compared in respect to the optimal solution.

Keywords: *point of interest, optimization, planning, Swap, Insert, Delete.*

1. Introduction

Tourists that visit one city or region during a trip of a limited time, find it impossible to visit all POIs that exist in that particular area. Thus, they have to select some POIs that they consider as more interesting and worthy for them. Doing the plan of visit that includes most interesting POIs to visit, for the available time, is usually a complex task. In such situations, it would be helpful for the tourist to have a system that runs on a hand held device, which would enable him to automatically plan the touristic trip. In general, systems like that tend to fulfill as much as possible the satisfaction of tourist by making a personalized trip plan. Usually, the constraints considered by planning systems under discussion are: geographical locations of POIs and their opening and closing hours, personal score of each POI for the tourist, duration of the trip, etc. In order to produce a trip that fits all/most of these constraints, a heuristic that tends to find an optimized trip needs to be introduced.

The simplest problem in trip planning can be compared to the Orienteering Problem (OP) [1], where a number of *n* locations are given, each of them having a score *s*. The goal is to have a single tour trip that includes as many points as possible, so the satisfaction factor of the trip is maximized. The Team Orienteering Problem (TOP) is an extended form of OP, which generalizes the problem for multiple tours [2]. Further, the Team Orienteering Problem with Time Windows (TOPTW) is an advanced version of TOP, where each location is associated with a time window that represents the timings when the visit could be realized [3]. In TOPTW the goal is to determine *m* routes, each limited by T_{max}, that maximizes the total collected score. In fact, the TOPTW is a simplified version of the Tourist Trip Design Problem (TTDP) [4].

An additional feature associated to these planning systems is that they have to plan the trip in real time, so that they can respond to changing user requests and preferences as well as unexpected events. In order to achieve optimal values, for such hard planning problems, for the execution time of around tens of seconds, we need to use a meta-heuristic to solve the problem. For instance, when more time than planned, is spent to visit a particular POI, tourists would like to be able to generate an updated trip plan in real time (possibly not lasting more than some tens of seconds).

The main contribution of this paper is introduction of an algorithm that finds optimal solutions for trip planning problem, in the execution time of some tens of seconds (varying from the particular details of the trip). The goal is to find a solid solution (not the best possible) in less than 10 seconds. This can be achieved by having a fast evaluation process of the candidate solutions, and by utilizing flexible operators to explore the search space. In this paper, we use taboo search heuristic as a guiding method for the search process. The taboo search heuristic uses the memory lists to save the prior search information, which, afterwards, is used to guide the search process towards finding the global optimal solution.

In the next section a literature review is presented and in Section 3 a problem definition is introduced. In Section 4 the detailed description of the algorithm is given, while in Section 5 the experimental results are shown. Discussions, conclusions and future work are elaborated in Section 6 and 7 respectively.

2. Literature review

There are many algorithms that deal with OP, TOP or TOPTW that are discussed in the literature. These algorithms could be also applied for the purpose of trip planning. A taboo search heuristic that effectively solves the TOP is presented in [5]. Roughly, this heuristic finds the solution by iteratively repeating the steps of initialization, solution improvement and evaluation. It utilizes a number of input parameters, which are used to fine tuning the performance of the algorithm.

Another algorithm, which is based on Guided Local Search (GLS) method [7] and can solve the TOP problem as well, is elaborated in [6]. In consecutive iterations of the algorithm, GLS method does the penalization of specific unwanted solutions. The penalization operator decreases the value of evaluation function for the specific solutions. This enables the algorithm to escape from getting stuck in the local optimum and carry on with further searches in different regions of search space.

An algorithm that is based on Variable Neighborhood Search (VNS) method [8], is presented in [9]. VNS systematically searches for a better solution, by changing the procedure of neighborhood creation. This changes the search direction either towards the optimal solution (local search) or towards the opposite direction (shaking of search process). This method, in its basic version, has the advantage of not requiring too many input parameters, while being able to produce solutions of high quality.

In [10] a simple algorithm that belongs to the family of Iterated Local Search (ILS) [11] is presented. Based on the experimental results, this algorithm is able of finding good solutions when applied for the data sets known in the literature. In a combined way, the operators of *Insertion* and *Shake* are applied. Instead of finding a number of random ILS solutions, the algorithm does build a sequence of solutions, obtained with the local search method. In this algorithm, it is important to have a balance between the number of iterations of algorithm execution and the frequency of *Shake* operator utilization.

Greedy Randomized Adaptive Search Procedure (GRASP) [12] is first used to solve the TOP in [13]. In general, GRASP method is executed for a pre specified number of iterations, where initially the procedure for solution construction is executed, followed by a procedure for local search. The behavior of the procedure for solution construction is controlled by the so called parameter “Greediness”, which represents the quotient between the Greediness and Randomness of the algorithm. This quotient shows how much the algorithm uses Greediness approach compared to Randomness approach, or vice versa. Different iterations of the algorithm are independent from each other, which mean that they return independent results.

In a Tourist Information System (TIS) presented in [14], authors use a trip planning algorithm that is developed using genetic algorithms. The path for visiting the selected POIs is created in two separate steps. The first step does the calculation of the shortest path between each and every POI, by using A* algorithm. The second step decides about the order of visits to particular POIs. In this case, the genetic algorithms are used to create a list of candidate solutions. Furthermore, regardless of the execution time of the algorithm, an approximated solution is always returned. By using genetic algorithms, the algorithm under discussion, is able to propose multiple paths to the tourists. This flexible feature will allow them to select one of the proposed routes.

3. Formulation as mathematical problem

The proposed algorithm lies on the field of optimizations of touristic tours, where a number of constraints are considered for planning and optimization of the tour. The goal is to plan a multiple day trip that will serve the tourist to visit a number of touristic sites/POI (Point Of Interests). This problem can be considered as a version of Orienteering Problem with Time Windows (OPTW). A set of n locations is given, where each of them $(i=1,\ldots, n)$ is associated with a satisfaction value S_i ,an entrance fee f_i, a typical visit duration *ti* an opening *(Oi)* and closing *(Ci)* hour. Usually the trip has several tours, with breaks in between (night time and mid daybreaks). Each tour is limited to a maximal period of time *Tmax* and it starts at a particular fixed point and ends at another fixed point. In general, the starting / ending time and tour duration are variable for each particular day. Mostly, the starting and ending point are the same for a tour of a single day (e.g. the tour starts and finishes at the hotel). The time *tij* needed to travel from location *i* to *j*, and vice versa, is known for all locations. In general, not all locations can be visited during the trip, since the duration of the trip is limited to m tours and the tours themselves are limited to T_{max}. Each location can be visited at most once. The visit is associated with a maximum budget B_{max}, which should not be exceeded throughout the entire trip.

No waiting times are considered at the POIs, meaning that the tourist will not have to wait any time prior to the realization of the visit to the POIs. This determines an additional constraint that makes sure that the timings of visits v_i are scheduled only when POIs are open.

The aim is to find a trip with m tours that includes as many available POIs as possible, by ensuring that the trip remains under budget and also taking in to account the total satisfaction factor and travel time of the trip. The intention is to have a higher satisfaction factor and shorter travel time. The trip budget and duration is considered as hard constraint, while the constraints of satisfaction factor and travel time are taken as soft constraints, and as such take part in the evaluation of the proposed solution.

The constraints of satisfaction factor and travel distance are non proportional between themselves. For instance, if there are more POIs in the trip, the satisfaction factor will be higher, while the travel time will be higher too, which conflicts with the travel time aspiration constraint. On the other hand, in order to degrease the travel time, it is needed to have less visits in the trip, which would minimize the satisfaction factor, which again opposes the intention to get a maximal satisfaction factor for the trip. Hence, defining an evaluation function of the trip that enables finding the optimal value, of both satisfaction factor and travel time constraints, is needed.

Based on the facts elaborated above, the running planning problem can be defined with following mathematical expressions:

$$Max\{\textstyle\sum_{i=1}^{n}(S_i * x_i)\} \qquad (1)$$

$$\textstyle\sum_{i=1}^{n}(B_i * x_i) \le Bmax \qquad (2)$$

Where:

$$x_i = \begin{cases} 1, & if\ point\ \boldsymbol{i}\ is\ visited\ during\ the\ trip \\ 0, & if\ point\ \boldsymbol{i}\ is\ not\ visited\ during\ the\ trip \end{cases}$$

n – Number of available POIs for visit
S_i – Satisfaction factor of point ***i***
B_i- Entry fee of point ***i***

$$Min\left\{\left[\sum_{i=1}^{n}\left(\sum_{\substack{j=1\\ j\neq i}}^{n} t_{ij} * y_{ij}\right)\right] + \sum_{i=1}^{n}(t_{si}\,u_i + t_{ie}v_i)\right\} \qquad (3)$$

Where:

$$y_{ij} = \begin{cases} 1, & if\ a\ visit\ to\ \boldsymbol{i}\ is\ followed\ by\ a\ visit\ to\ \boldsymbol{j} \\ 0, & if\ a\ visit\ to\ \boldsymbol{i}\ is\ not\ followed\ by\ a\ visit\ to\ j \end{cases}$$

$$u_i = \begin{cases} 1, & if\ visit\ \boldsymbol{i}\ is\ first\ visit\ in\ the\ tour \\ 0, & if\ visit\ \boldsymbol{i}\ is\ not\ first\ visit\ in\ the\ tour \end{cases}$$

$$v_i = \begin{cases} 1, & if\ visit\ \boldsymbol{i}\ is\ last\ visit\ in\ the\ tour \\ 0, & if\ visit\ \boldsymbol{i}\ is\ not\ last\ visit\ in\ the\ tour \end{cases}$$

t_{ij} – travel time from point ***i*** to ***j***
t_{si} – travel time from start point to point ***i***
t_{ei} – travel time from point ***i*** to end point

$$\textstyle\sum_{i=1}^{m} z_{ij} \le 1 \qquad (j=1, \ldots, n) \qquad (4)$$

Where:

$$z_{ij} = \begin{cases} 1, & if\ visit\ \boldsymbol{i}\ in\ trip\ is\ point\ \boldsymbol{j} \\ 0, & if\ visit\ \boldsymbol{i}\ in\ trip\ is\ not\ point\ \boldsymbol{j} \end{cases}$$

m – Number of POIs visited during the entire trip

$$o_i \le v_i\ \ \&\ \ v_i + t_i \le c_{i,}\ \ i = 1, \ldots, m \qquad (5)$$

Expression (1) defines the intended maximal satisfaction factor of the trip, while expression (2) ensures that trip is equal or lower than the budget allocated for the trip. Formula (3) expresses the minimal travel time aspiration, by considering the travel times between visited points themselves and also between them and the starting/ending points. Expression (4) makes sure that a particular point is visited at most one time, while expression (5) makes the trip feasible only when all the points of interests are open on their scheduled time.

4. Description of the algorithm

Overview

Trip planning is done based on the entry data that describe the trip. The entry data sets are categorized in three different kinds:

- Data that describe the trip, such as: start/end date of trip, allocated budget, accommodation location, and number of tours to be taken during the trip, coefficient of weight of satisfaction factor and travel time, tourist preferences for categories and types of POIs and an additional entry parameter that specifies one of the two possible regimes of work of the algorithm.
- Data that describe the POIs, such as: name of the POI, typical visit duration, location, entry fee, working hours, type and category of POI.
- Data about travel distances between each and every POI that is available. The travel distances are expressed in unit of minutes.

As seen in the figure 1, the trip planning algorithm consists of two separate modules. One of them deals with calculation of personal score for the POIs, while the other one does the actual planning of the trip. In this paper, calculation of personal score (satisfaction factor), which is

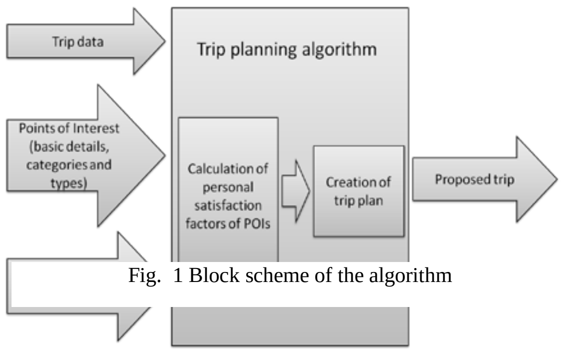
Trip data
Points of Interest (basic details, categories and types)
Trip planning algorithm
Calculation of personal satisfaction factors of POIs
Creation of trip plan
Proposed trip

Fig. 1 Block scheme of the algorithm

the process that is known as matchmaking between tourist preferences and POIs, is done by utilizing a simple algorithm introduced by Souffriau & Maervoet et al [15]. The value range of satisfaction factor produced by this matchmaking algorithm is between 0 and 48. Since, in our case we assume that the range of values for satisfaction factors of POIs is between 0 and 100, we have used a transformation function to convert the range of values from *[0 – 48]* to the range *[0 – 100]*.

$$Satisfaction\ factor_{(Max=100)} = \frac{100 * [Satisfaction\ factor_{(Max=48)}]}{48}$$

Module for trip planning

The process of trip planning creates an itinerary that consists of predefined number touristic tours to be taken during the trip period. The optimization of the trip planning is done by utilizing the taboo search heuristic. In our case, the taboo search heuristic uses the operators of Swapping and Insertion for exploring the search space. The Delete operator is used in some iteration to escape the local optimum. The taboo search heuristic is known for its process of memorizing previous search information, which facilitates the escape from local optimum by changing the search direction. In our example, the planning module can be customized by nine different entry parameters, as shown in the pseudo code given below.

```
Algorithm Main(TLS, MT, MTWI, MBTNTS, ACI, PC, FMH, DN, FTWMV)
begin
    Operators = {Swap, Insert};
    Initialize taboo memories;
    Create initial solution $S_c$;
    Evaluate $S_c$;
    $S_b = S_c$;
    iterationNumber = 0; IterationsWithoutImprovement = 0;
    while (iterationNumber <= MT) do
        Divert=(iterationsWithoutImprovement % DN) == 0;
        for each operator in Operators do
            Generate neighbourhood of $S_c$ by using current operator;
            Find best non taboo and taboo neighbour of $S_c$ (Divert);
            if IterationsWithoutImprovement greater than ACI then
                AspirationCriteria=best taboo nighbor >
                best solution found so far;
            else
                AspirationCriteria= best taboo nighbor –
                best non taboo nighbor > MBTNTS;
            end
            if there is a feasible non taboo / taboo neighbour then
                if AspirationCriteria is fulfilled then
                    $S_c$ =Best taboo neighbour;
                else
                    $S_c$ =Best non taboo neighbour;
                end
                if operator is Swap then
                    acceptanceCriteria = $S_c$ better than $S_b$;
                else
                    if FTWMV then
                        acceptanceCriteria= $S_c$ better than $S_b$ or
                        number of visits in $S_c$ >number of visits in $S_b$ ;
                    else
                        acceptanceCriteria= $S_c$ better than $S_b$;
                    end
                end
                if acceptanceCriteria is fulfilled than
                    $S_b = S_c$;
                end
            else
                Delete a visit from trip;
            end
        next;
        if there is improvement in current iteration then
            IterationsWithoutImprovement=0;
        else
            IterationsWithoutImprovement +1;
        end
        if IterationsWithoutImprovement equals MTWI then
            Exit loop;
        end
        iterationNumber +1;
    end
end
Return $S_b$;
```

In the Table 1 we present the description for the entry parameters of the algorithm.

Table 1: Algorithm parameter description

Parameter	***Abbrev.***	***Description***
Taboo List Size	TLS	Specifies the number of iterations that a move will remain taboo. E.g. *TLS=5* indicates that swapping between point ***i*** and ***j*** cannot be performed in next five iterations.
Max Tries	MT	Indicates the total number of iterations that the algorithm will run.

Max Tries Without Improve-ment	MTWI	Specifies the total number of iterations that the algorithm will run without any further improvement.
Margin Between Taboo And Non Taboo Solution	MBTNTS	Indicates how much better a taboo solution should be, compared to a non taboo solution, such that it would fulfill the aspiration criteria.
Aspiration Criteria Iterations	ACI	Defines the number of iterations without improvement, which will utilize the version of aspiration criteria with margin between taboo and non taboo solution. After passing the number of iterations, indicated by ACI, the aspiration criteria is calculated in its usual form (accepting a taboo solution only if it is better than the best solution found that far).
Penalty Coefficient	PC	Takes a value between 0 and 1, which is used to penalize frequent moves that have occurred during the search process.
Frequency Memory Horizon	FMH	Determines the number of iterations after which the frequency based memory will be reset.
Diversifica-tion Number	DN	Specifies how often the search process will be diversified. Every DN iterations the diversification process will take place.
Find Trip With Maximum Visits	FTWMV	It is a logical parameter that defines one of the two possible regimes of work of the algorithm. If its vale is *True*, the algorithm will try to find the best trip with maximal number of POIs. Conversely, if its value is *False*, the algorithm will focus only in finding the best evaluated trip, even though the resulting trip may not have the maximal number of POIs.

The algorithm uses two operators for exploring the search space, which are shown in the initial part of the pseudo code. The Swap operator does the swapping of POIs that are on trip with POIs that are currently out of the trip. Insertion of new POIs into the trip is made by Insert operator. The so called Taboo Memories are used to save information about the recency and frequency of swaping and inserting individual POIs. These memories will enable the search process to avoid getting stuck in the local optimum and also direct the search process in the new regions of search space (that far not explored). Before the algorithm starts looping, an initial solution is created, which is than evaluated and accepted as best current solution. In general, the initial solution is created by randomly inserting new POIs, until there is no left space.

The algorithm will be iteratively executed by MT iterations. The Boolean variable *Divert* will be calculated for each iteration of the algorithm and it is used to decide whether the search diversification operator shall be applied in current iteration. Its value is *True* if division of variable *iterationsWithoutImprovement* and parameter *DN* returns an integer, otherwise its value is *False*.

Inside the main algorithm loop, another loop (named the *operator loop*) is executed for two times. In the first execution, the *operator loop* uses the Swap operator, while in the second time it uses the Insert operator. In both executions, with Swap and Insert operator, the generation of neighborhood is full, which means that all possible combinations are considered. Swap operator swaps each POI on trip with each POI out of the trip, while the Insert operator inserts each non included POI before and after each included POI.

After the process of neighborhood generation, each valid neighbor is evaluated and the best non taboo and neighbor of current iteration are selected. In case the evaluation is done for the neighborhood generated by Swap operator, for some specific iterations (exactly every DN iterations) the operator of penalizations is used.

After finding the best two solutions (one of them taboo and the other one non taboo), the algorithm checks whether the aspiration criteria is fulfilled. This algorithm, depending on the value of ACI parameters, works with two sorts of aspiration criteria. If value of variable *IterationsWithoutImprovement* is greater than value of parameter *ACI*, then the aspiration criteria is defined as: "*Best taboo neighbor must be better than best solutions found so far, so that the taboo neighbor could be accepted as actual solution*", otherwise, the aspiration criteria is defined as "*result of subtraction between taboo and non taboo solution should be greater than the value of parameter MBTNTS, so that the best taboo solution is accepted as actual solution*".

If at least one of the neighbor solutions (taboo or non taboo) represents a feasible solution, the algorithm carries on with selection of the aspiration criteria, otherwise, the operator that deletes a POI from the trip, is applied. The POI deletion is conducted randomly in one of the tours of the trip. If aspiration criteria is fulfilled, then best taboo neighbor is accepted as the actual solution, otherwise the best non taboo solution is accepted as current solution.

Next, the variable *acceptanceCriteria* is defined, which is used to determine whether the current solution could be accepted as best solution found so far. Depending on the value of logical parameter FTWMV, the variable *acceptanceCriteria* could be defined in two different ways. If its value is *False*, than the varable *acceptanceCriteria* will be set to allow a current solution to become the best solution found so far, only if it is better. Otherwise, when parameter *FTWMV* is *True*, a current solution can become the best solution if it has more visit on the trip, regardless that it may not have a greater evaluation then the best solution found so far.

Inside the algorithm loop, the number of iterations without improvement is counted. If this number reaches the value defined by parameter MTWI or the predefined number of maximum iterations MT exceeds, than the algorithm execution stops and the best found solution is returned.

Determining the legality of the neighbor

A neighbor would be legal if it fulfills the hard constraints:

- All visits in the trip are scheduled when respective POIs are open,
- The trip budget is not exceeded, and
- The length of each tour in the trip remains in the pre specified duration

The pseudo code for determining candidate feasibility is given in the following:

```
Determine legality of neighbor
begin
    legality=false;
    if  new vist is open in scheduled time do
        if neighbor cost is under budget do
            if neighbor is viable in time do
                legality= true;
            end
        end
    end
end
return legality;

Determine time viability of neighbor(changed tour)
begin
    viability=false;
    if length of changed tour is not grater than orginla tour length do
        vilabilty=true;
    end
end
Return viabilty;
```

Evaluation function

Evaluation of the candidate solution is done by considering two soft constraints, namely *the total trip satisfaction factor* and *total trip travel time*. The goal is to find an optimal trip that has the total satisfaction factor as higher as possible, while the travel time remains as low as possible. In order to realize this, we have used an evaluation/fitness function that consists of two components:

$$Evaluation\ function = w1 * [\ satisfaction\ factor_{norm}\ ] + w2 * [travel\ time_{norm}\ ]$$

Parameters $w1$ and $w2$ represent the weight coefficients for the particular components of the evaluation function. In order to have a proportional effect in to the evaluation function, when the value of individual components changes, we have used the normalized values of both components:

$$satisfaction\ factor_{norm} = 100 * \frac{total\ satisfaction\ factor}{maximal\ satisfaction\ factor}$$

Where:

$$total\ satisfaction\ factor = \sum_{i=1}^{n} SF_i$$

$$maximal\ satisfaction\ factor = 100 * \frac{\text{MNP}}{\text{SDT}} * [\text{TDWB}\ ]$$

SF_i–Satisfaction factor of POI with index i,
n – Number of POIs included into the trip,
MNP – Maximal Number of POIs that are aimed to be visited per day
SDT – Standard Duration of a Tour
TDWB – Trip Duration Without Breaks

Based on the practical experience, we consider that the maximal desired number of POIs to be visited during one day tour is 20, while the duration of the tour of one day is usually 8 hours. In order to have a realistic view for the maximal satisfaction factor, we have considered only the time when the tourist is supposed to be active in his trip (TDWB), by omitting the breaks that the tourist may take (e.g. such as sleeping at the hotel at night).

Since we use the approach of maximizing the value of evaluation function, mathematically, we would need to maximize the values of both its components. While for the satisfaction factor component this is right, for the travel time component it should be the opposite aim. Hence, in order to facilitate the maximization of both components and aim in minimizing the travel time, we try to maximize the complementary value of travel time, which in fact will minimize the travel time, by using:

$$travel\ time_{norm} = 100 * (1 - \frac{travel\ time}{\text{TDWB}})$$

5. Experimental results

The algorithm is tested by utilizing 10 different instances of tourist profiles. In addition, we have used 40 instances of POIs of the city of Vienna. As a starting/ending point of each tour of the trip we have used a hotel in the same city. Travel distances between POIs are expressed in the unit of minute.
All calculations are made by using a PC with an Intel Core 2 processor with 2.0 GHz and the RAM memory of 2.55 GB.

In the following experiments, if not differently stated, we have used a trip with execution details as shown in table 2.

Table 2: Default data for experiments

Parameter	***Value***
Trip duration	Two tours, five hours each
Trip Budget	200 euro
Tour start time	11:00
Tour end time	16:00
Weight of satisfaction factor	70%
Weight of travel time	30%
Execution time of the algorithm	5 minutes

Our experiments aim in obtaining the optimal values for the entry parameters of the algorithm, such as: finding the optimal taboo list size, margin of aspiration criteria, frequency of applying the operator for search diversification etc. If not differently stated, the algorithm is executed 10 times for each instance, and then, the average values of the results of particular executions are taken.

Tests with various versions of initial solutions

We have tested the algorithm with three different kinds of versions of initial solutions:

1. *Random initial solution* – where POIs are randomly entered into the trip itinerary, as much as there is room in it.
2. *Initial solution with POIs sorted in ascending or*der – where POIs are entered into the trip based on the value of satisfaction factor. The POIs that have lower satisfaction factor are prioritized for earlier insertion into the trip itinerary.
3. *Initial solution with POIs sorted in descending order* –in this case, as well as in the previous case, the POIs are entered into the trip based on the value of satisfaction factor. Conversely, in this case the POIs that have higher satisfaction factor have higher chance for earlier insertion into the trip itinerary.

By using instance 8, the algorithm is executed 10 times for each three different initial solutions. Respective results of the execution of the algorithm for each different initial solution are compared, and then the maximal value from one of the three initial solutions is recorded. The number of maximums shown in figure 2, indicate that random initial solution performs better than the other two initial solutions, because it has been better in seven executions compared to the other initial solutions. On the other hand, the ordered lists versions (both in ascending and descending order) have never resulted better than the other ones. In three executions, at least two of the three different initial solutions have produced the same evaluation of the produced solution.

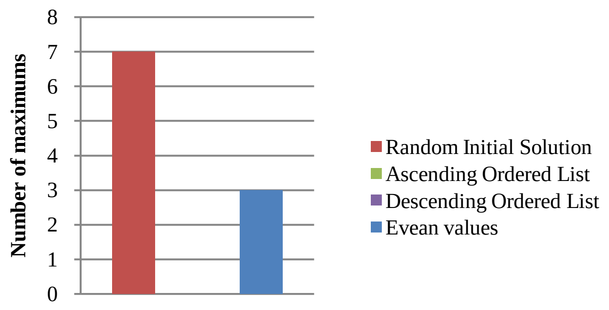

Fig. 2 Algorithm performance for different initial solutions

Variance of the algorithm result for different executions

Instance 8 is executed 10 times and the variance between different executions is shown in the following figure.

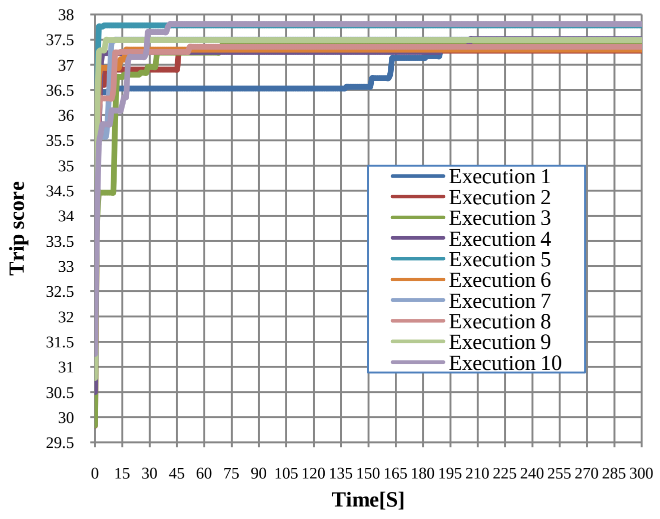

Fig. 3 Variance of the algorithm performance for different executions

It can be stated that for almost all executions, the best solution is found for approximately 40 seconds (except execution 1), and the solution with average evaluation is found for approximately 10 seconds.

Selection of taboo list size

In this experiment, all instances are executed 10 times with taboo list sizes 3, 6 and 9. Afterwards, the average values of individual executions of instances are calculated. Then the average values for individual taboo list sizes for all instances are taken. The results are shown in the below table.

Table 3 : Comparison of taboo list size

Taboo list size	Minimum	Maximum	Average	Standard deviation
3	31,376	31,599	31,444	0,072
6	31,680	31,947	**31,769**	0,084
9	31,392	31,510	31,430	0,041

Furthermore, we have also counted the number of instances for which a particular taboo list size produces better results (cf. figure 4).

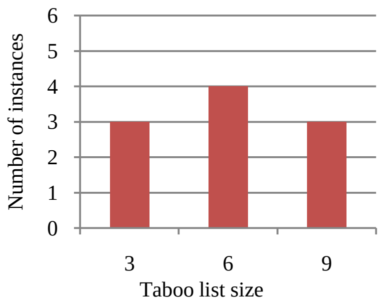

Fig. 4 Selection of taboo list size

From table 3 and figure 4, it can be conclude that taboo list size of 6 produces better results. In general, the solutions obtained by using the taboo list size of 6, are better for average 0.3 points than the solutions gained by two other taboo list sizes used in the experiment.

Diversification of search process

The diversification process ensures that the algorithm continues to search for the global optimal solution. This process is applied every *N* iterations. The experiment shows that applying the search diversification process yields to better results. Furthermore, if we apply it more often, we would gain better results.

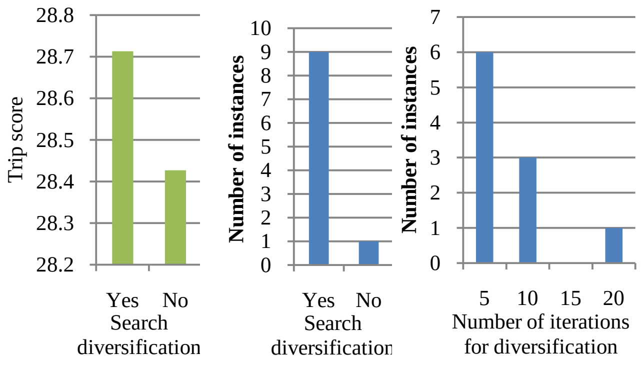

Figure 5 Advantages in applying search diversification process

Best trip versus trip with maximum POIs

The algorithm under discussion works in two different kinds of modes. The first one tries to find the highest evaluating trip, while the second one, aims in finding the best evaluating trip that has maximum number of POIs. The working mode of the algorithm is specified by the user.
The following figure, expresses a comparison between the two algorithm regimes in terms of execution time, number of POIs and evaluation.

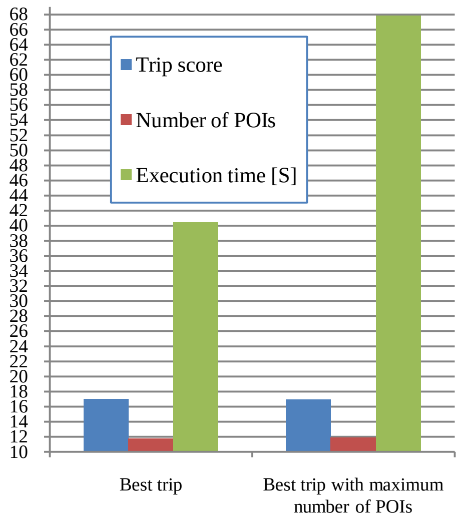

Fig. 6 Comparison of the two regimes of the algorithm

As seen in the figure, the overall trip score and number of POIs do not have a significant increase in the second mode (Best trip with maximum POIs) compared to the first mode (Best trip). Conversely, it only increases the average execution time for around 28 seconds.

Comparison of algorithm results for different tourist instances

In figure 7, we show the variation of trip score for different tourist instances. It can be noticed that for all instances, for the period of around 10 seconds, we gain solutions that evaluate near to final solutions. During this 10 second period, almost all instances are improved for about three points compared to their initial solution. After this period, no significant improvements are made (in most cases the improvement is less than one point). Hence, it can be concluded that further execution of the algorithm does not bring to significant improvement. Conversely, it will only have the negative impact of increasing the execution time of the algorithm.

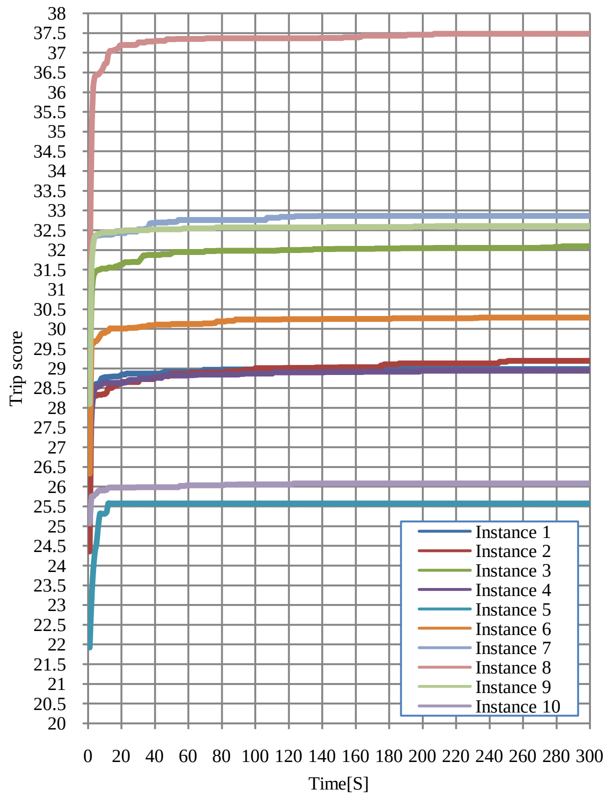

Fig. 7 Execution of the algorithm for different tourist instances

Comparison of different implementation of Swap operator

The basic implementation of Swap operator swaps each POI that is on the trip itinerary with each POI outside the trip itinerary. We have called this as “Large Swap”, since the solution neighborhood is created with all possible combinations enabled by Swap operator. In addition, by using Min/Max Conflicts method, we have implemented the Swap operator in its “Small Swap” mode, where only three POIs of the current trip itinerary that have the largest travel time (travelling time from previous POI to the current POI) are considered for swapping with the POIs out of the trip itinerary. The third version of the Swap operator is implemented by using the Hill climbing method.

In figure 8, we have shown the execution of instance no. 8 with the three different versions of Swap operator. The instance no. 8 is executed 10 times with each different implementation of Swap operator, and the average values of 10 executions are presented in the figure.

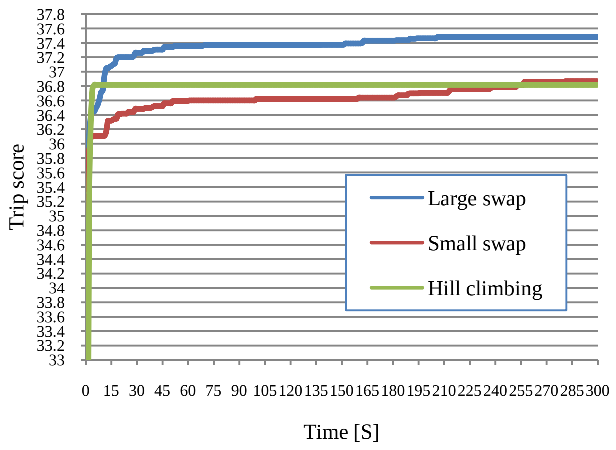

Fig. 8 Performance of the algorithm for different implementation of Swap operator

Figure 8 shows that the “Large Swap” version yields to better results, while the “Small Swap” version and the Hill Climbing method evaluate nearly to the same value. The Hill Climbing method performs faster than the other two versions (best solution is found in around 5 seconds), but quality of the solutions found by this method is worse. In addition, “Large swap” version is quicker (best solution found in approximately 210 seconds) than the “Small swap” version (best solution found in about 260 seconds).

Comparison of different implementation of Swap operator for various number of tours

In table 4, we show results gained by executing the algorithm (using instance No. 8) with different Swap operator implementation and different trip lengths.

For small number of tours (one or two tours), the “Large Swap” mode performs better than the other two implementations of Swap operator. It is noticeable that for a short duration of the trip (one or two tours), the “Large Swap” version takes only about 10 seconds more than the “Small Swap” version. On the other hand, for larger trips (three or more tours) the “Small Swap” version is quicker for about 50 seconds. Furthermore, when the trip consists of five tours, the “Small Swap” version is faster for around 130 seconds than the “Large Swap” version. Considering these results, sometimes it may be more appropriate to sacrifice a little bit in the quality of the found solution (by using the “Small Swap” version), in order generate the trip plan faster. The Hill Climbing method does not take so much time to find the final solution (in average 86 seconds), but the quality of the found solutions and the small number of POIs on those solutions, makes this method as not successful as the other two methods.

Table 4: Performance of the algorithm for different trip lengths

	Large Swap			***Small Swap***			***Hill Climbing***		
Number of tours	Trip score	Number of visits	Time [S]	Trip score	Number of visits	Time [S]	Trip score	Number of visits	Time [S]
1	37,4	4	53,6	37,35	4	52	35,15	2,6	40,2
2	35,7	8	87,9	35,41	7,9	67,9	35,47	6,7	80,7
3	34,5	11,8	127,8	33,86	11,4	81,7	34,07	10	97
4	33,4	15,5	168,1	32,76	14,3	73,3	32,86	13,2	105,6
5	32,0	18,1	226,2	31,70	17,4	96,3	31,82	15,7	109,1
Average	34,63	11,48	132,7	34,22	11,00	74,2	33,88	9,64	86,5

6. Discussions

In this paper we presented an algorithm that is used for planning the touristic trip, by considering a number of soft and hard constraints. The hard constraints consist of opening and closing hours of POIs, the trip budget and duration. The solutions generated by the algorithm are evaluated by using a fitness function that considers the overall trip satisfaction factor and tourist travel time throughout the entire trip, which in fact represent the soft constraints for the algorithm. The calculation of personal satisfaction factors for the POIs is done by using a simple algorithm introduced by [15]. The algorithm is created by using the taboo search metaheuristic, where four different kinds of initial solutions are tested. The exploration of search space is done by using the operators of Insertion, Swapping and Deletion. In order to test the performance of the algorithm, the Swap operator is implemented in three different formats. First two implementations are done by using the small and large Swap approach, respectively, while the third one is done by using the Hill Climbing method. In each iteration of the algorithm, the Insert operator tries to insert a POI in one of the available tours. The Delete operator is applied in occasional iterations, so would let the algorithm to escape from getting stack in an endless loop.

Algorithm performance test is done by conducting a number of experiments, which are mainly realized to obtain the optimal values of the entry parameters of the algorithm. The experiment with the initial solution shows that random initial solutions perform slightly better than the other ones. In addition, it is obvious that the variance between the results of different executions of the algorithm is less than one. The optimal number of iterations for which a solution would remain taboo is six. In general, solutions gained when using the taboo list size of six, score for 0.3 points more than when the taboo list size is three or nine.

It is evident that the utilization of search diversification process yields to better results. In the conduced experiments, we notice an average improvement of 0.3 points when diversification is applied. Furthermore, experimental results show that the more often we apply the diversification, the better results we gain. The penalty coefficient of 0.8 has a slight advantage in comparison to the other tested values.

Depending on the working mode of the algorithm, finding the best trip or the best trip with maximal POIs, will take approximately an average time of 40 or 60 seconds, respectively. The quality of found solutions in both regimes is nearly the same.

In terms of quality, the experiments with different implementation of Swap operator show that the “Large Swap” version outperforms the other two versions. The “Large Swap” version scores better than the “Small Swap” version and the Hill climbing method for 0.7 and 0.8 points, respectively. The Hill climbing method is able to find the final solutions in about 5 seconds, whereas the “Large and Small Swap” need much more time, which may be up to 200 or 250 seconds, respectively.

The experiments with different trip lengths show that for a trip of one or two tours, it may be more appropriate to use the “Large Swap” mode, since the quality of the solutions is better, whereas the execution time remains nearly the same to that of “Small Swap” mode. In addition, for larger number of tours (3 or more), it may be acceptable to sacrifice a little bit the quality of solutions, so that we could gain the final solution quicker by using the “Small Swap” mode.

Finally, based on the experimental results, the algorithm is able to produce a personal trip itinerary in margins of tens of seconds. Further, in order to meet specific requirements, the algorithm can be configured by using nine different parameters. The presented results indicate that for a reasonable time of execution, the algorithm generates a near to optimal trip plan.

7. Conclusions and future work

The main contribution of this paper is the introduction of an algorithm for touristic trip planning that is comparable to the well known problem of Team Orienteering Problem with Time Windows (TOPTW). In general, the trip planning can be done in an average time of 70 seconds. The solution evaluation is made by using a fitness function that consists of two separate components, where one of them considers the overall trip satisfaction factor and the other one the total traveling time. A such fitness function makes the trip plan more personal for the tourist and the algorithm suitable for use in personal trip planning systems. The algorithm performance is tested by using 40 instances of POIs of the city of Vienna and 10 different tourist profiles. The future work includes testing the algorithm with larger test instances. Additionally, testing the algorithm with test dates known in the literature will make it comparable to the existing similar algorithms. It may also be important to design new and more specific test instances, for example concerning the number of possible visits, number of tours, and the length of the time windows of POIs etc.

The relative long time to finding the optimal solution that mainly comes as the result of the process of verifying the legality of proposed solutions, may be a focus of research of work in the future. Furthermore, adding new planning constraints such as, context factors (weather, unexpected events, traffic jams, weekends etc.) could lead to more personalized trip plans.

In the real life, it often happens that a group of tourists go for a joint touristic trip. Hence, introduction of an algorithm that is able to plan a trip for group of tourists may be desired. The consideration of personal interests of individual tourists would be preferable. It would be ideal, if the algorithm could create a master trip (for the whole group) that in some portions of it could be spread into some sub trips, so that it would match interests of sub groups of tourist, who may have different preferences for specific POIs. The evaluation of the trip would need to be a general one, for the whole group of tourists, by considering a number of soft and hard constraints concerning the touristic trip.

References

[1] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. Naval Research, Logistics, 34:307-318, 1987.

[2] Chao, I.-M., B. L. Golden, E. A. Wasil. 1996b. The team orienteering problem, Eur. J. Oper. Res. 88(3) 464–474.

[3] Kantor, M. G., M. B. Rosenwein. 1992. The orienteering problem with time windows. J. Oper. Res. Soc. 43(6) 629–635.

[4] P. Vansteenwegen, D. Van Oudheusden. The mobile tourist guide: an OR opportunity. OR Insights 2007; 20(3):21-7.

[5] Tang, H., Miller-Hooks, E.: A Taboo search heuristic for the team orienteering problem. Comput. Oper. Res. 32, 1379 – 1407 (2005)

[6] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, D. Van Oudheusden, D.: A guided local search metaheuristic for the team orienteering problem. Eur. J. Oper. Res. 196(1), 118-127 (2008). Doi: 10.1016/j.ejor.2008.02.037

[7] Voudouris, C., Tsang, E.: Guided local search and its application to the travelling salesman problem. Eur. J. Oper. Res. 113, 469-499 (1999).

[8] Hansen, P., Mladenovic, N.: Variable neighbourhood search: Principles and applications, Eur. J. Oper. Res. 130, 449-467 (2001).

[9] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, D. Van Oudheusden, D.: Metahuristics for Trip Planning. Metaheuristics in the Service Industry, pages:15-31, 10.1007/978-3-642-00939-6_2, (2009).

[10] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, D. Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows, Journal of Computers & Operations Research, 36 (2009) 3281 -3290.

[11] H.R. Lourenço, O. Martin, and T. St¨utzle, "Iterated local search," in Handbook of Metaheuristics, ser. International Series in Operations Research & Management Science, F. Glover and G. Kochenberger, Eds., Kluwer Academic Publishers, vol. 57, pp. 321–353, 2002.

[12]Feo, T.A., Resende, M.G.C. : A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 867-71, (1989).

[13]Souffriau, W., Vansteenwegen, P., Berghe, G.V., Oudheusden, D.V. : A greedy randomised adaptive search procedure for the Team Orienteering Problem, EU/MEeting 2008 on metaheuristics for logistics and vehicle routing location, (2008)

[14] Maruyama, A., Shibata, N., Murat,a Y., Yasumoto, K., and Ito, M. (2004). A personal Tourism Navigation Sstem to Support Traveling Multiple Destinations with Time Restrictions. Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA '04), IEEE (2004)

[15] Souffriau, W., Maervoet, J., Vansteenwegen, P., Berghe, G.V., Oudheusden, D.V. : A mobile tourist decision support system for small footprint devices, IWANN 2009, Part I, LNCS 5517, pp. 1248-1255, (2009)

First Author: Kadri Sylejmani, Dipl. Ing. in Computers and Telecommunication – 2004, Msc. in Computer Science – 2010; Teaching Assistant at Faculty of Electrical and Computer Engineering – Department of Computer Engineering, University of Prishtina, Kosovo; has presented several papers in scientific conferences and workshops on his field of research; his current research interest include filed of electronic tourism and problem solving in Artificial Intelligence.

Second Author: Agni Dika, PhD in Computer Science – 1989; Full professor at Faculty of Electrical and Computer Engineering – Department of Computer Engineering, University of Prishtina, Kosovo; his current research interest include computer logic design and algorithms.