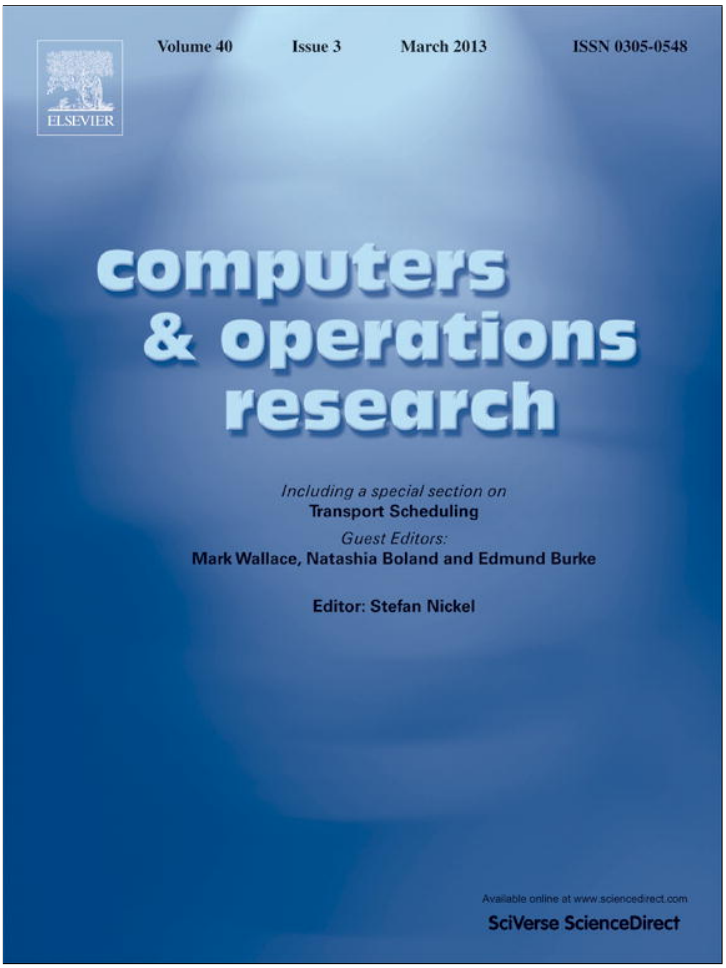

Computers & Operations Research 40 (2013) 758–774

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Integrating public transportation in personalised electronic tourist guides

Ander Garcia [a,*], Pieter Vansteenwegen [b], Olatz Arbelaitz [c], Wouter Souffriau [d], Maria Teresa Linaza [a]

[a] Department of eTourism and Cultural Heritage, Vicomtech, Paseo Mikeletegi 57, E-20009 Donostia, San Sebastian, Spain
[b] Department of Industrial Management, Ghent University, Belgium
[c] Department of Computer Architecture and Technology, University of the Basque Country, Spain
[d] Centre for Industrial Management, Katholieke Universiteit Leuven, Belgium

ARTICLE INFO

Available online 13 April 2011

Keywords:
Time-Dependent team orienteering problem with time windows
Public transportation
Tourist guide

ABSTRACT

Personalised electronic tourist guides (PETs) are mobile hand-held devices able to create tourist routes matching tourists' preferences. Transportation information has been identified as one of the most appreciated functionalities of a PET. We model the tourist planning problem, integrating public transportation, as the time-dependent team orienteering problem with time windows (TD-TOPTW) in order to allow PETs to create personalised tourist routes in real-time. We develop and compare two different approaches to solve the TD-TOPTW. Experimental results for the city of San Sebastian show that both approaches are able to obtain routes in real-time.

1. Introduction

Many tourists enter a local tourist office (LTO) to look for information about the destination they are visiting. They want to compose a trip based on their personal interests and up-to-date point of interest (POI) information. The problem a tourist faces is to decide which POIs to visit or to filter and select what activities to do [1]. The next step is to time-sequence these POIs or activities and to decide how to move from one POI to the following. The LTO staff helps the tourists with this time consuming task, categorising their profile and interests (cultural, romantic, science, etc.) and determining their restrictions (time, money, etc.). Combining this information with their knowledge about local POIs (prices, timetables, etc.), the staff suggests a personalised route to the tourists.

Obviously, this route does not take into account new circumstances that may occur during the visit, such as spending more time than expected at a POI. In such cases, tourists can try to update their personal route based on the information available or follow the more obvious tourist paths.

A personalised electronic tourist guide (PET), which is a mobile hand-held device, can perform at least the same task that is now fulfilled by the LTO staff: it should create personalised routes that maximise the tourists' satisfaction, taking into account several restrictions, such as opening hours, duration of the visits, entrance fees and travel distances. Since tourists do not expect to wait minutes to receive a new route, a PET should provide an integrated solution for POI selection and route planning that adapts to new circumstances in real-time. Furthermore, transportation information should be integrated, since that is identified as one of the most appreciated functionalities of a PET [2–4].

The requirements of the optimisation problem a PET has to solve can be modelled by the tourist trip design problem (TTDP) [5–7]. The orienteering problem (OP) is the simplest version of the TTDP. In the OP [8], several POIs with an associated score can be visited in order to obtain a total trip score. A tourist can visit each POI only once. The objective is to select (and order) the set of POIs that maximise the total trip score without violating a given time restriction. The team orienteering problem with time windows (TOPTW) has been successfully used to model the optimisation problem of a PET, without the integration of public transportation [9,10].

In this paper, we present a new extension of the OP that allows integrating the use of public transportation by the PET: the time-dependent team orienteering problem with time windows (TD-TOPTW). A typical TD-TOPTW will contain a number of POIs with a fixed location, opening hours (time windows) and a given score. Movements between POIs can be done on foot or by using public transportation. The public transportation network consists of a number of fixed stops and different lines between these stops, each with a given frequency.

In the OP and TOPTW, the travel time between the POIs is fixed. In the TD-TOPTW, the travel time between POIs will vary, because a tourist can choose between walking and using public transportation. Furthermore, the waiting time for a public service depends on the moment the

* Corresponding author. Tel.: +34 943309230; fax: +34 943309393.
E-mail addresses: agarcia@vicomtech.org (A. Garcia), pieter.vansteenwegen@UGent.be (P. Vansteenwegen), olatz.arbelaitz@ehu.es (O. Arbelaitz), wouter.souffriau@dynavic.com (W. Souffriau), mtlinaza@vicomtech.org (M.T. Linaza).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.03.020

tourist arrives at the boarding point. Thus, the travel time between two POIs depends on the leave time of the first POI and the transportation mode. This changing travel time significantly increases the difficulty of the problem, especially since computations have to be done in real-time.

We present and compare two different approaches to solve the TD-TOPTW. Both algorithms are applied on a set of test instances based on real data for the city of San Sebastian. The main contribution of this research is that we developed two approaches that solve the TD-TOPTW in real-time.

This paper is organised as follows. In the next section, we review existing literature. In Section 3 we describe the structure of the algorithm used to solve the TD-TOPTW. In Sections 4 and 5, we introduce two different approaches to tackle routing problems with a public transportation network. In Section 6 we indicate the configuration of some algorithm parameters. In Section 7, we show the experimental results. Conclusions and further work are discussed in Section 8.

2. State-of-the-art

This section briefly reviews the state-of-the-art of personalised tourist guides, the orienteering problem and planning applications for public transportation.

As power capabilities, network technologies and portability of mobile devices have improved, several applications aiming at creating electronic tourist guides have been developed. They have been called mobile tourist guides (MTG), personal navigation systems for tourism (PNS), electronic tourist guides (ETG) or personalised electronic tourist guides (PET). Vansteenwegen [11] presents an extensive review of existing PETs. However, none of these guides apply advanced heuristics to solve the planning problem nor integrate the use of public transportation. Dynamic tour guide [12] is the most advanced and mature PET found in the literature. Although it created and recalculated routes in real-time (less than 5 s), the routing algorithm was very simple and it had important restrictions: it could only create proper solutions for 1 day routes and a small number of POIs. A PNS called P-tour [13] applied a genetic algorithm to calculate routes. Nevertheless, tourists had to manually enter the POIs they wanted to visit with their details (visiting time, duration and value). Moreover, the system needed nearly 10 s to obtain a route for just 12 POIs. Again, no public transportation was taken into account. Zenker et al. [14] presented ROSE, a mobile application assisting pedestrians to find preferred events and comfortable public transport connections composed by three main services: recommendation, route generation and navigation. They identified the route planning problem to solve and they described it as a multiple destination recommendation with public transportation and time windows. This is the same problem as the TD-TOPTW. However, they did not find any algorithm able to solve it.

The simplest version of the TD-TOPTW is the OP [8]. Its generalisation to a multiple-day trip is known as the team orienteering problem (TOP). Vansteenwegen et al. [15] present an extensive review of these problems, solution algorithms and applications. When POIs have an associated time window, the problem is called TOP with time windows (TOPTW) [16]. Righini and Salani [17] applied bi-directional dynamic programming to solve the orienteering problem with time windows (OPTW) instances to optimality. Regarding the TOPTW, Tricoire et al. [18] presented a variable neighbourhood search algorithm for a generalisation of the TOPTW called multi-period OP with multiple time windows (MuPOPTW). Vansteenwegen et al. [10] developed an efficient metaheuristic to solve the TOPTW. Montemanni and Gambardella [19] proposed using ant colony systems for the TOPTW. Interested readers can find a thorough revision of literature about OPTW and TOPTW in these papers.

Fomin and Lingas [20] present the time-dependent OP (TDOP), an extension of the OP where the time needed to travel from a POI i to a POI j depends on the leave time from i. Implicitly, they can deal with different transportation modes. However, they do not present an algorithm that can be used in real-time applications and no time windows or multiple-day trips are considered.

The model we propose is the time-dependent TOPTW (TD-TOPTW), which combines the previous models and makes it possible to integrate one or more public transportation networks to travel between POIs. The travel time required to move from one POI to the next will vary according to the transportation mode (walking or public transportation) and the leave time. To the authors' understanding, there is no algorithm able to solve the TD-TOPTW.

Although examples are present of finding the shortest path in public transportation networks or time-dependent networks, they all focus on solving individual queries. Pyrga et al. [21] provide extensive information about the earliest arrival problem (EAP). They have worked on timetable information problems within railway transportation systems. An EAP query (A, B, t_0) consists of a departure station A, an arrival station B, and a leave time t_0. The main objective is to find a connection between A and B leaving not earlier than t_0 and arriving as soon as possible. Ding et al. [22] focused on finding the best departure time to minimise the total travel time on time-dependent graphs. A large number of public transportation providers implement these algorithms. For example, the public transportation service provider of San Sebastian has a best path finding application to move between two points (http://www.dbus.es/es/usuarios/planificador-rutas).

As examples of more advanced applications, Zografos and Androutsopoulos [23] implemented an algorithm to calculate optimum itineraries in Athens's urban public transport system, including a stop at an intermediate point. In Hong-Kong, a similar system was tested using a different approach [24]. Within the tourism field, PECITAS, a mobile personal navigation system to find the best path between two POIs regarding both the travel time and tourist's preferences was proposed by Tumas and Ricci [25]. All these applications focus on POI-to-POI cheapest cost problems, while this paper focuses on a higher level selection of and routing problem between multiple POIs.

3. Solution strategy

The solution strategy we design to tackle the TD-TOPTW is based on the algorithm proposed by Vansteenwegen et al. [10] for the TOPTW. This algorithm is the best algorithm available to obtain high quality results for the TOPTW in real-time [15]. This makes it also promising for dealing with the TD-TOPTW in real-time. In this section, we give a general description of the TOPTW algorithm. For more details we refer to the TOPTW article.

The heuristic is based on iterated local search (ILS) [26]. ILS is a metaheuristic method based on iteratively building sequences of solutions generated by an embedded heuristic called local search. This leads to much better solutions than repeating random trials of the same heuristic. The heuristic perturbs the solution found by the local search to create a new solution. Then, it takes the best solution as the new starting solution for the local search. The process is repeated until a termination criteria is met. The ILS metaheuristic can be summarised as in Algorithm 1.

Algorithm 1. Diagram of iterated local search.

```
s0 = GenerateInitialSolution;
s* = LocalSearch(s0);
While termination condition NOT met do
    s' = Perturbation(s*);
    s*' = LocalSearch(s');
    s* = AcceptanceCriterion(s*,s*');
```

The local search heuristic inserts new visits to a route, one by one. For each visit i that can be inserted, the cheapest insertion time ($Shift_i$) is determined. For each of these visits the heuristic calculates a ratio, which relates the score of the POI to the time required to visit it. Among them, the heuristic selects the one with the highest ratio for insertion. This process is repeated until no more POIs can be inserted. The perturbation phase removes consecutive POIs from a route. After the removal, the heuristic shifts all visits following the removed visits towards the beginning of the route as much as possible, in order to avoid unnecessary waiting. Until a termination criterium is met, the perturbation procedure and the local search heuristic are executed. Finally, the heuristic returns the incumbent solution as the result. A summary of the heuristic is shown in Algorithm 2.

Algorithm 2. ILS heuristic.

```
PositionStartRemove = 1;
NumberToRemove = 1;
NumberOfTimesNoImprovement = 0;
maxNumberToRemove = NumberOfPOIs/(3* NumberOfDays);
MaxIter = factorNoImprovement * Size of 1st route;
While NumberOfTimesNoImprovement < MaxIter do
    While not local optimum do
        Insert;
    If Solution better than BestFound then
        BestFound = Solution;
        NumberToRemove = 1;
        NumberOfTimesNoImprovement = 0;
    else
        NumberOfTimesNoImprovement + 1;
    Shake Solution (NumberToRemove,PositionStartRemove);
    PositionStartRemove = PositionStartRemove + NumberToRemove;
    NumberToRemove + 1;
    PositionStartRemove = PositionStartRemove mod Size of smallest Route;
    If NumberToRemove equals maxNumberToRemove then
        NumberToRemove = 1;
return BestFound
```

This method needs to be adapted to solve TD-TOPTW instances. The envisioned application needs a fast and effective algorithm. During every iteration of the algorithm, many evaluations of possible insertions are considered, before the most promising POI is actually inserted. In order to obtain a fast algorithm, it is required that the evaluation of possible insertions only involves the POI that is inserted and the two POIs between which the new POI will be inserted. This requirement is a consequence of the conclusion by Vansteenwegen et al. [10] that it is too time consuming to check the time window of each POI in the trip, every time an insertion is considered. Moreover, the evaluation of a possible insertion becomes much more complex when public transportation is included. A small delay in the planned leave time from one POI can cause a significant delay in the arrival time at the next POI. For instance, when a tourist just misses the bus and has to wait for the next one or walk to the next POI instead. For the TD-TOPTW, this implies that, when considering an insertion, not enough calculation time is available to check the (possibly required) changes in travel time and transportation mode between all other visits in the trip. Even when an entire time-dependent travel time matrix is calculated beforehand, recalculating the entire trip (including possible changes in travel times) cannot be done fast enough to do this every time a POI is considered for insertion. Therefore, similar to the algorithm proposed by Vansteenwegen et al. [10], we developed a way to evaluate possible insertions locally, only involving the POI that is inserted and the two POIs between which the new POI is inserted.

We propose two different approaches in order to handle the public transportation difficulty. The first one involves a precalculation step (Section 4), where we calculate the average travel times between all pairs of POIs. With these average travel times, we solve the TD-TOPTW as a regular TOPTW. A repair procedure adapts the arrival and leave times of the visits of the resulting trip according to the differences between the average travel times and the (time dependent) real travel times. As a consequence, one or more visits can become unfeasible and it may be necessary to remove them before proposing the trip to a tourist.

The second approach is based on a modified and fast evaluation of the possible insertions (Section 5). By introducing a few new concepts, it becomes possible to evaluate each insertion locally and efficiently. Since the ILS heuristic evaluates many possible insertions, we argued above that a highly efficient method is required to achieve high quality results in real-time.

The main contribution of our research is that we designed two solution approaches for the TD-TOPTW that have almost the same performance as the fastest algorithm for the TOPTW, based on quality of results and computational cost.

4. Average travel time

This approach to tackle the integration of public transportation is based on precalculating the average travel time for each pair of POIs. The motivation for this approach is twofold. Firstly, this precalculation does not have the real-time requirement, secondly, due to the high frequency public transportation service, the average travel time is rather accurate in practice.

We calculate the travel times between all the POIs with time steps of 1 min. Within this calculation we take into account all the possible transportation modes. Then, we obtain an average travel time for each pair of POIs. This calculation is done only once and then we store all the results in a database.

As this step does not have a real-time requirement, any suitable algorithm can be used. We have implemented a time-dependent Dijkstra's shortest path algorithm aiming at public transportation to calculate the shortest path between the origin and destination POI, at each departure time [27].

Certainly, faster and more suitable algorithms than Dijkstra's algorithm are available to calculate these travel times. Besides the methods discussed in our state-of-the-art (Section 2), an overview of latest speed-up techniques for transportation networks can be found in Delling et al. [28] and Bauer et al. [29]. Moreover, traffic assignment algorithms [30,31] could also be applied to calculate the average travel times. However, how these travel times are calculated is not the key point of our algorithm. We applied Dijkstra's algorithm as a proof of concept of our average travel time approach.

In order to limit the number of possibilities for each pair of POIs, it is possible to limit the transfer distance between stops of different lines, the maximum walking distance between POIs and the maximum distance between POIs and stops. Once the average travel times are available, we can solve the TD-TOPTW as a regular TOPTW. A repair procedure introduces the real travel times between the POIs of the final TOPTW solution. If the real travel time is smaller than the average one, only the travel time is adapted by advancing the visit towards the beginning of the route as much as possible. Otherwise, if the real travel time is larger than the average one, some visits will have to start later. If this causes a visit to become unfeasible, we remove it from the route and we move the rest of the route forward.

One way to evaluate this approach is to verify how many POIs are removed to restore feasibility. For the test instances presented in this paper (Section 7), removals are only required for one of the 28 instances.

5. Real calculation

In the second approach, we design some concepts that allow to make a very fast local evaluation of each possible insertion. A local evaluation should only involve the POI that is inserted and the two POIs between which the new POI is inserted. In order to simplify the explanation, we will just use "bus", instead of using "public transportation", in the rest of this section. Nevertheless, this approach can also be applied for trains, subways or any other public transportation mode.

We introduce $maxDelay_i$ to indicate the maximum time a departure from a POI (l_i) can be delayed in order not to make any visit of the route unfeasible. Based on *maxDelay* of the POI before which the new POI is inserted, a local evaluation of the insertion is possible.

We create a model, considering the periodicity of the bus services, where $maxDelay_i$ can be calculated and updated in an efficient way. This model is limited to direct bus connections, having no transfers between lines. At the end of this section, we extend this model to include transfers in public transportation, either precalculating some required values or modelling transfers as direct connections. In this way, transfers can be taken into account as well. In many cities, it is valid to assume a periodic timetable and a fixed frequency for each bus line. Furthermore, it can be assumed that tourists will not make use of the earliest and the latest services, which might not have a fixed frequency.

Before we explain how $maxDelay_i$ should be calculated, we need to introduce some more concepts: $Wait_i$, the $MaximumArrivalDelay_i$, $MaxTrans_i$, $minTrans_i$ and $transPeriod_i$. If the tourists arrive at a POI i (a_i) before the opening of the time window (O_i), they will have to wait before starting the visit:

$$Wait_i = \max(0, O_i - a_i) \tag{1}$$

The $MaximumArrivalDelay_i$ (MAD_i) indicates how much the currently scheduled arrival time at a certain POI can be delayed without making any visit unfeasible. It is nothing more than the sum of $Wait_i$ and $maxDelay_i$:

$$MAD_i = Wait_i + maxDelay_i \tag{2}$$

$MaxTrans_i$ represents the maximum time the leave time of a visit can be delayed without changing the transportation mode between POIs i and $i+1$. This change in transportation mode can be between walking and taking the bus or vice versa, but it can also be between taking this bus and the next bus.

The second one is its counterpart, $minTrans_i$. $minTrans_i$ represents the maximum time the leave time can be decreased without changing the transportation type between POIs i and $i+1$. The third one is $transPeriod_i$ and it represents the period of the public transportation service travelling between POIs i and $i+1$. When a bus line has a high frequency (e.g. 4 per hour), the bus will have a small period (e.g. $transPeriod_i = 15$ min).

5.1. Three travel scenarios

In order to explain how these concepts are used, we differentiate between three travel scenarios, between each pair of POIs. Each scenario will require a proper local evaluation of possible insertions:

- *Always walk*: If no bus connection is available between a pair of POIs i and $i+1$, or walking is always faster than taking a bus, the travel time between POIs i and $i+1$ will always be the walking time, no matter what the leave time is at POI i. Fig. 1 shows the travel time as a function of the leave time at POI i. Fig. 2 shows the arrival time at POI $i+1$ as a function of the leave time at POI i. In this scenario, *MaxTrans* and *minTrans* are infinite and *transPeriod* is not applicable. *wT* indicates the walking time between the POIs.

- *Always bus*: If the walking distance between two POIs i and $i+1$ is longer than a predefined distance, the system will always propose to take the bus. This also applies to the case when the sum of the maximum waiting time for the bus and the bus travel time is smaller than the walking time. The travel time between two POIs by taking the bus is the sum of the real bus travel time (*busTime*) and a waiting time before the bus arrives. This waiting time is limited by the period of the bus. If there is a bus every 15 min, the maximum waiting time will also be 15 min, in the case the tourists just missed the previous bus. Thus, the total bus travel time varies between a minimum value (*busTime*) and a maximum value (*busTime*+*transPeriod*). Obviously, the time required to walk from POI i to the boarding point of the bus and from the arrival point to POI $i+1$ should be included in the *busTime* (*bT*) as well. Fig. 3 shows the travel time as a function of the leave time at POI i. Fig. 4 shows the arrival time at POI $i+1$ as a function of the leave time at POI i. It can be observed in Fig. 3 that the total travel time decreases with the same amount as the leave time increases, unless the bus is missed and the tourists have to wait for the next bus. As a consequence, the arrival time does not change when the leave time is delayed, as long as the bus is not missed (Fig. 4). Otherwise, it is increased by the period of the service.
- *Bus or walking*: In this scenario, it depends on the exact leave time at POI i what the fastest transportation mode will be walking (when there is a long waiting time for the bus – Fig. 5, j) or taking the bus (when the waiting time is short enough – Fig. 5, k). The travel time has a maximum value equal to the walking time (*wT*) and a minimum time equal to the *busTime* (*bT*).
  The arrival time is a mix of the two previous scenarios: if the tourists wait for the bus (Fig. 6, k), they could leave later and still arrive at the same time (with the same bus). However, if they leave too late they will miss the bus and they will go on foot (Fig. 6, j).

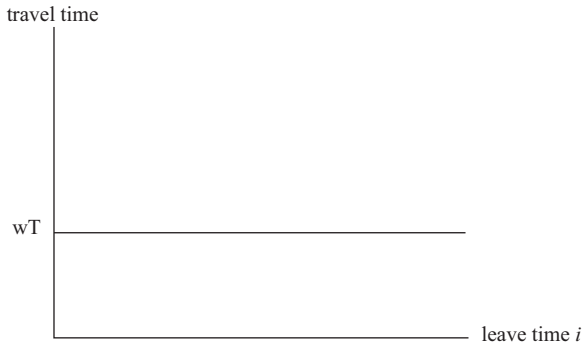

Fig. 1. Travel time always walk.

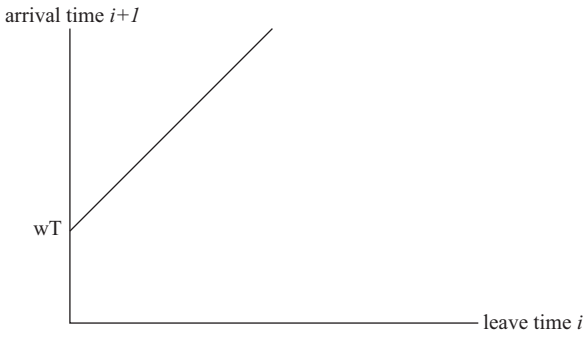

Fig. 2. Arrival time always walk.

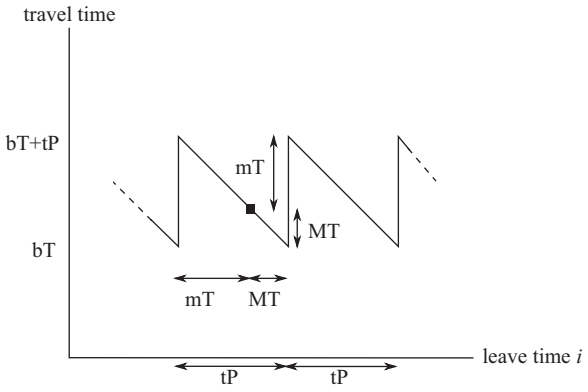

Fig. 3. Travel time going by bus.

5.2. Three types of travel time changes

Knowing the current transportation scenario, the current leave time from POI i to POI $i+1$ and the related *MaxTrans*, *minTrans* and *transPeriod*, there are three types of changes to the travel time that need to be considered for updating these concepts, and making a local evaluation possible.

The first one consists on knowing how much time earlier (t') the tourists arrive to POI $i+1$ if they leave from POI i t time units earlier. This query is equivalent to moving left on the leave-arrival time graphs (Figs. 2, 4 and 6). We have called this change advance departure, AD(t). Algorithm 3 shows the updating formulas that have to be applied in each of the three abovementioned scenarios. In these formulas, "*t*%*transPeriod*" stands for "*t modulo transPeriod*".

The formulas for scenario 1 and 2 are straightforward, since the transportation mode never changes. For scenario 2, the only question that remains is whether the waiting time for the bus has increased or decreased due to the change in leave time. The formulas for scenario 3 are illustrated with an example in Fig. 7. In this example, a bus is available every 10 min (departure at 2, 12, 22, etc.) and the driving time is 7 min. The walking time equals 13 min. Thus, a tourist leaving in the interval (2,6) will walk and within [6,12] the tourist will take the bus. This cycle is repeated every 10 min (*transPeriod*). Currently, the tourist leaves at 25, goes on foot and arrives at 38. *MaxTrans* equals 1 and *minTrans* equals 3.

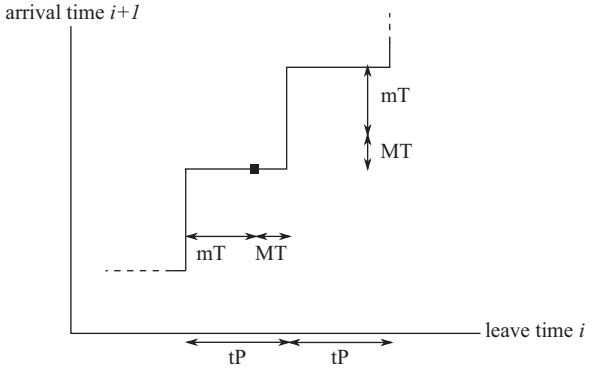

Fig. 4. Arrival time going by bus.

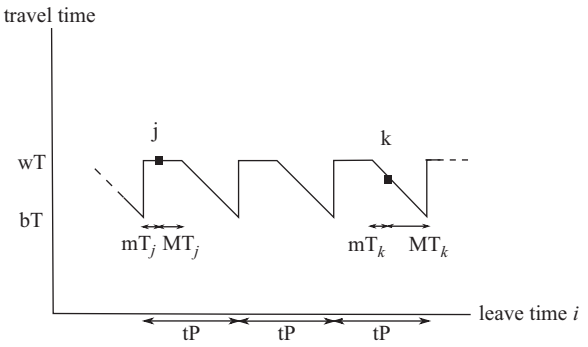

Fig. 5. Travel time going on foot and by bus.

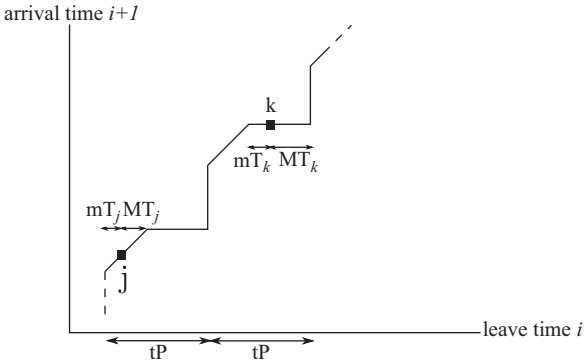

Fig. 6. Arrival time going on foot and by bus.

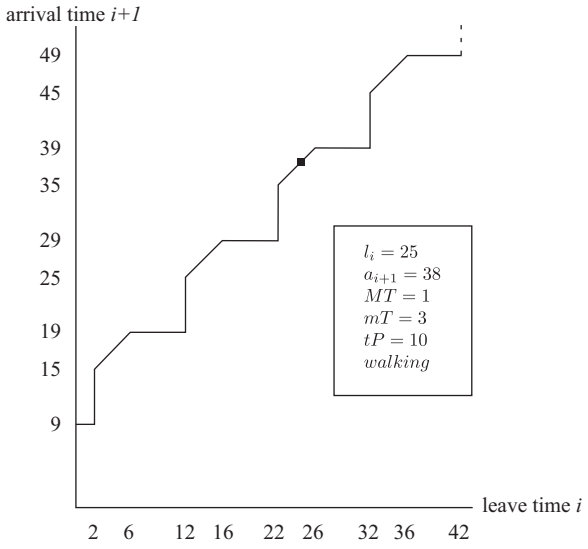

Fig. 7. Arrival time example for scenario 3.

Algorithm 3. AD updates.

scenario 1: *always walk*;
t′ = t;
scenario 2: *always bus*;
if *t*%*transPeriod* < *minTrans* **then**
  t′ = t − t%transPeriod;
  MaxTrans′ = MaxTrans + t%transPeriod;
  minTrans′ = minTrans − t%transPeriod;
else
  t′ = t + (transPeriod − t%transPeriod);
  MaxTrans′ = MaxTrans − (transPeriod − t%transPeriod);
  minTrans′ = minTrans + (transPeriod − t%transPeriod);
scenario 3: *bus or walking, when current transportation is walking*;
if *t*%*transPeriod* < *minTrans* **then**
  t′ = t;
  MaxTrans′ = MaxTrans + t%transPeriod;
  minTrans′ = minTrans − t%transPeriod;
else if *minTrans* < *t*%*transPeriod* < (*transPeriod*−*MaxTrans*) **then**
  t′ = t − t%transPeriod + (transPeriod − MaxTrans);
  MaxTrans′ = t%transPeriod − minTrans;
  minTrans ′ = (transPeriod − MaxTrans) − t%transPeriod;
else
  t′ = t;
  MaxTrans′ = MaxTrans−(transPeriod−t%transPeriod);
  minTrans′ = minTrans + (transPeriod−t%transPeriod);
scenario 3: *bus or walking, when current transportation is bus*;
if *t*%*transPeriod* < *minTrans* **then**
  t′ = t − t%transPeriod;
  MaxTrans′ = MaxTrans + t%transPeriod;
  minTrans′ = minTrans − t%transPeriod;
elseif *minTrans* < *t*%*transPeriod* < (*transPeriod*−*MaxTrans*) **then**
  t′ = t + minTrans;
  MaxTrans′ = t%transPeriod − minTrans;
  minTrans′ = (transPeriod − MaxTrans) − t%transPeriod;
else
  t′ = t + transPeriod − t%transPeriod;
  MaxTrans′ = MaxTrans − (transPeriod − t%transPeriod);
  minTrans′ = minTrans + (transPeriod − t%transPeriod);

If the visit to the POI is advanced 15 min (leave at 10), the arrival will be 19 (=15−15%10+(10−1)) minutes earlier. The new *MaxTrans*′ equals 2 (=15%10−3) and the new *minTrans*′ equals 4 (=10−1−15%10). In this case the bus should be used. Thus, the travel time would be reduced 4 (=10−1−15%10) min, arriving 19 min earlier instead of 15. If the tourist leaves 19 min earlier (leave at 6), the arrival will be 19 min earlier, at 19. The new *MaxTrans*′ equals 0 (=1−(10−19%10)) and the new *minTrans*′ equals 4 (=3+(10−19%10)). In this case, the tourist will be walking.

The second change consists on calculating how much time later (t') the tourists arrive to POI $i+1$ if they leave POI i t time units later. This is equivalent to moving right on the leave-arrival time graphs (Figs. 2, 4 and 6). We have called this change delay departure, DD(t). Algorithm 4 shows the formulas that have to be applied. Again, the formulas for scenarios 1 and 2 are straightforward while the formulas for scenario 3 can be verified based on the example of Fig. 7.

The third change consists on knowing how much time later (t') the tourists can leave from POI i to arrive to POI $i+1$ not later than t time units. This query is equivalent to moving up the arrival at POI $i+1$ on the leave-arrival time graphs (Figs. 2, 4 and 6). We have called this change delay arrival, DA(t). Algorithm 5 shows the formulas that have to be applied for each scenario.

Algorithm 4. DD updates.

scenario 1: *always walk*;
t′ = t;
scenario 2: *always bus*;
if *t*%*transPeriod* < *maxTrans* **then**
  t′ = t − t%transPeriod;
  MaxTrans′ = MaxTrans − t%transPeriod;
  minTrans′ = minTrans + t%transPeriod;
else
  t′ = t + (transPeriod − t%transPeriod);
  MaxTrans′ = MaxTrans + (transPeriod − t%transPeriod);
  minTrans′ = minTrans − (transPeriod − t%transPeriod);
scenario 3: *bus or walking, when current transportation is walking*;
if *t*%*transPeriod* < *MaxTrans* **then**
  t′ = t;
  MaxTrans′ = MaxTrans − t%transPeriod;
  minTrans′ = minTrans + t%transPeriod;
else if *MaxTrans* < *t*%*transPeriod* < (*transPeriod*−*minTrans*) **then**
  t′ = t − (t%transPeriod − MaxTrans);
  MaxTrans′ = (transPeriod − minTrans) − t%transPeriod;
  minTrans ′ = t%transPeriod − MaxTrans;
else
  t′ = t;
  MaxTrans′ = MaxTrans + (transPeriod−t%transPeriod);
  minTrans′ = minTrans−(transPeriod−t%transPeriod);
scenario 3: *bus or walking, when current transportation is bus*;
if *t*%*transPeriod* < *MaxTrans* **then**
  t′ = t − t%transPeriod;
  MaxTrans′ = MaxTrans − t%transPeriod;
  minTrans′ = minTrans + t%transPeriod;
else if *MaxTrans* < *t*%*transPeriod* < (*transPeriod*−*minTrans*) **then**
  t′ = t − minTrans;
  MaxTrans′ = (transPeriod − minTrans) − t%transPeriod;
  minTrans′ = t%transPeriod − MaxTrans;
else
  t′ = t + (transPeriod − t%transPeriod);
  MaxTrans′ = MaxTrans + (transPeriod − t%transPeriod);
  minTrans′ = minTrans − (transPeriod − t%transPeriod);

Algorithm 5. DA updates.

scenario 1: *always walk*;
t′ = t;
scenario 2: *always bus*;
t′ = t−t%transPeriod + MaxTrans;

scenario 3: *bus or walking, when current transportation is walking*;
if *MaxTrans* < *t*%*transPeriod* < (*transPeriod*−*minTrans*) **then**
  t′ = t + (transPeriod − minTrans) − t%transPeriod;
else
  t′ = t;
scenario 3: *bus or walking, when current transportation is bus*;
if *t*%*transPeriod* > (*MaxTrans*+*minTrans*) **then**
  t′ = t − minTrans;
else
  t′ = t − t%transPeriod + MaxTrans;

5.3. Local evaluation of an insertion

Based on the concepts we introduced and the formulas to keep the concepts updated, it is now possible to make a very fast and local evaluation of each possible insertion.

A visit to POI j can be inserted between POIs i and k if the extra time required by the insertion ($Shift_j$) (formula 3) is smaller than the sum of $Wait_k$ and $maxDelay_k$ (formula 4). In order to determine the visit that will be selected for insertion, first, we determine the lowest possible $Shift_j$ for each possible extra visit, i.e. the best possible insert position. Then, for each visit, we calculate the same ratio as in the TOPTW algorithm [10]. Finally, we select the visit with the highest ratio for insertion. T_j represents the duration of the visit to POI j, and c_{ij} is the travel time between POIs i and j:

$$Shift_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik} \tag{3}$$

$$Shift_j \leq Wait_k + maxDelay_k \tag{4}$$

$$Ratio_j = (Score_j)^2 / Shift_j \tag{5}$$

Furthermore, the changes considered above allow updating easily all POI visits when a POI is inserted or removed.

When a POI is inserted, the visits POIs after the insertion need to be delayed. Starting from the POI after the inserted POI, all the POIs are delayed until the extra time required by the insertion is absorbed by reducing waiting or travel times.

The following formulas are used to update the visit k when visit j is inserted between i and k. a_k represents the arrival time at POI k, and l_k is the leave time:

$$Wait'_k = \max[0, Wait_k - Shift_j] \tag{6}$$

$$a'_k = a_k + Shift_j \tag{7}$$

$$Shift_k = \max[0, Shift_j - Wait_k] \tag{8}$$

$$l'_k = l_k + Shift_k \tag{9}$$

$$maxDelay'_k = maxDelay_k - Shift_k \tag{10}$$

In order to calculate how much time later the tourists arrive at $k+1$ when they leave $Shift_k$ time units later from k, we make use of the DD change introduced previously. Next to updating *MaxTrans* and *minTrans* from k to $k+1$, the following formulas are used to update the visits after k, one after another, until *Shift* is reduced to zero:

$$Wait'_{k+1} = \max[0, Wait_{k+1} - DD(Shift_k)] \tag{11}$$

$$a'_{k+1} = a_{k+1} + DD(Shift_k) \tag{12}$$

$$Shift_{k+1} = \max[0, DD(Shift_k) - Wait_{k+1}] \tag{13}$$

$$l'_{k+1} = l_{k+1} + Shift_{k+1} \tag{14}$$

$$maxDelay'_{k+1} = maxDelay_{k+1} - Shift_{k+1} \tag{15}$$

Once *Shift* is reduced to zero, visits before the insertion may require an update of *maxDelay*. In order to calculate *maxDelay* of visit i, first we use the DA change to find the maximum time leaving i can be delayed not to arrive to $i+1$ with a delay higher than MAD_{i+1} ($MaximumArrivalDelay_{i+1}$). Then, we compare this time with the remaining time until the end of visit to i and the closing time of i. In this query, we do not have to update *MaxTrans* or *minTrans*, because we are not moving the visit.

$$MAD_{i+1} = Wait_{i+1} + maxDelay_{i+1} \tag{16}$$

$$maxDelay_i = \min[C_i - l_i, DA(MAD_{i+1})] \tag{17}$$

When we remove a POI from a route, the visits to the POIs after it are moved towards the beginning of the route. Starting from the POI next to the removed one, all the POIs are moved forward until the free time obtained by the removal is consumed by waiting times or

extra travel times. The following formulas are used to update visit k when visit j is removed between i and k:

$$a'_k = l_i + c_{ik} \tag{18}$$

$$Wait'_k = \max[0, O_k - a'_k] \tag{19}$$

$$Shift_k = \max[0, a_k - a'_k - Wait'_k] \tag{20}$$

$$l'_k = l_k - Shift_k \tag{21}$$

Once visit k has been updated, the following formulas are then used to update the visits after k, one after another, until *Shift* is reduced to zero. In order to calculate how much time earlier the tourists arrive at $k+1$ when they leave $Shift_k$ time units earlier from k, we make use of the AD change introduced previously and we update the values of *MaxTrans* and *minTrans* from k to $k+1$:

$$a'_{k+1} = a_{k+1} - AD(Shift_k) \tag{22}$$

$$Wait'_{k+1} = \max[0, O_{k+1} - a'_{k+1}] \tag{23}$$

$$Shift_{k+1} = \max[0, a_{k+1} - a'_{k+1} - Wait'_{k+1}] \tag{24}$$

$$maxDelay'_{k+1} = maxDelay_{k+1} + Shift_{k+1} \tag{25}$$

Visits before the removal may require an update of *maxDelay*. Similar to inserting a POI, we make use of the DA change within this calculation.

Based on all these formulas, updating a trip after the actual insertion of a POI can be done faster than recalculating the whole trip. Notice that evaluating possible insertions is required many times every iteration (and, therefore, must be very fast), while actually inserting a POI, and updating the whole trip, only happens at most a few times every iteration.

5.4. Example

The next example summarises the different processes described in this section. In the example (Fig. 8) there are seven POIs: i, j, k, l, m, n and o. The duration of the visit of each POI is 5 min and the walking time between POIs and between POIs and bus stops is indicated in Fig. 8. An existing route is composed by visits $i-j-k-n-o$ and there are two candidate POIs to insert: l and m. There is one bus service with five stops represented by rectangles: 1, 2, 3, 4 and 5. The bus travel time between stops is 1 min. The first bus departs from stop 1 at 0 min and there is a service every 15 min.

Table 1 shows the details about the existing route. The columns represent the following data about each POI: opening time (*O*), closing time (*C*), arrival time (*a*), waiting time (*W*), leave time (*l*), *MaxTrans* (*MT*), *minTrans* (*mT*), *transPeriod* (*tP*), the type of transportation (*tt*, walk (*W*) or bus (*B*)), Delay Arrival (*DA*) and *maxDelay* (*mD*). The travel information between a POI i and the next one $i+1$ is represented on the row of POI i. Thus, the last POI does not have this data. *MaxDelay* has been calculated using formulas (16) and (17). DA has been calculated following Algorithm 5.

If we want to insert POI l or m between k and n, we have to check the feasibility of each insertion (formula 4):

$$Shift_l = c_{kl} + Wait_l + T_l + c_{ln} - c_{kn} = 10 + 0 + 5 + 12 - 12 = 15 \leq 18 \tag{26}$$

$$Shift_m = c_{km} + Wait_m + T_m + c_{mn} - c_{kn} = 20 + 0 + 5 + 10 - 12 = 23 \not\leq 18 \tag{27}$$

Since only $Shift_l$ is smaller than $maxDelay_n$, we insert POI l between k and n.

Table 2 shows the details about the new route $i-j-k-l-n-o$. We arrive to n $Shift_l$ (15) time units later, but due to previous $Wait_n$ (5), we leave n 10 time units later. Thus, arrival to o is delayed DD(10) = 11 time units, according to the Algorithm 4 for the third scenario when we go by bus. We calculate the new value of $maxDelay_o$ (3) and then we update *maxDelay* of the rest of the POI using formulas (2) and (17). Notice that due to the later arrival to n, the transportation mode from n to o has changed.

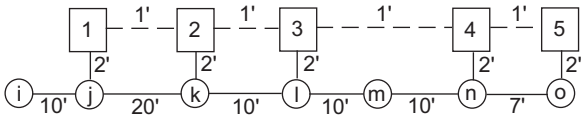

Fig. 8. Example.

Table 1
Details of initial route $i-j-k-n-o$.

POI	*O*	*C*	*a*	*W*	*l*	*MT*	*mT*	*tP*	*tt*	*DA*	*mD*
i	0	60	10	0	15	∞	∞	∞	*W*	10	10
j	0	40	25	0	30	13	2	15	*B*	28	10
k	0	80	48	0	53	6	9	15	*B*	21	21
n	70	100	65	5	75	1	1	15	*B*	13	13
o	70	100	81	0	86	–	–	–	–	–	14

Table 2
Details of the route after inserting POI *l*.

POI	*O*	*C*	*a*	*W*	*l*	*MT*	*mT*	*tP*	*tt*	*DA*	*mD*
i	0	60	10	0	15	∞	∞	∞	W	10	10
j	0	40	25	0	30	13	2	15	*B*	13	10
k	0	80	48	0	53	1	9	15	*W*	7	7
l	55	100	63	0	68	7	8	15	*B*	7	7
n	70	100	80	0	85	4	9	15	*W*	3	3
o	70	100	92	0	97	–	–	–	–	–	3

Table 3
Details of the route after removing POI *k*.

POI	*O*	*C*	*a*	*W*	*l*	*MT*	*mt*	*tp*	*tt*	*DA*	*mD*
i	0	60	10	0	15	∞	∞	∞	*W*	10	10
j	0	40	25	0	30	13	2	15	*B*	28	10
l	55	100	49	6	60	0	15	15	*B*	15	15
n	70	100	65	5	75	1	1	15	*B*	13	13
o	70	100	81	0	86	–	–	–	–	–	14

Finally, if we remove POI *k* from the route, visits after it ($l-n-o$) are moved forward. Table 3 shows the details of the route after updating all the values according to the formulas and algorithms presented during the previous section.

5.5. Modelling transfers

In the above mentioned formulas for AD, DD and DA we only considered direct connections between stops near origin and destination POIs. An argument for this simplification could be that, if two POIs are far from each other and no direct connection is available, it is very likely the algorithm will not schedule these POIs one immediately after the other. One or more other POIs will be scheduled in between and only direct public bus connections will be required. Still, it would be more realistic and it would probably lead to better solutions to model transfers as well.

In order to take transfers into account, extra travel scenarios, with transfers, should be considered during the calculation and update of advance departure (AD, Algorithm 3), delay departure (DD, Algorithm 4) and delay arrival (DA, Algorithm 5). When transfers are included, the key problem is determining the correct value of the waiting time at the transfer for all leave times.

The first approach we have implemented for dealing with transfers consists of precalculating the travel time, AD, DD and DA for each pair of POIs and for all required leave times and time shifts. But to precalculate the values for each pair of POIs and for each possible leave time and each possible shift is not an affordable option. For only 50 POIs and 8 h (480 min) days, this corresponds to ($50*50*480*480 =$)576 000 000 values, requiring more than a 1 GB of memory and our computer run out of memory during the calculation. To significantly reduce the size of the problem, we can benefit from the period (frequency) of the bus lines. For direct bus connections, the period of the service is known and it suffices to precalculate AD, DD and DA for each possible leave time in the period and each time shift smaller than the period. When transfers are involved, more complex calculations are required. The "common period" of the transfer connection will be the "least common multiple" of the periods of all individual bus lines involved in the transfer. AD, DD and DA should be precalculated for each possible leave time of the common period and for each shift smaller than the common period. Of course, every time an insertion is actually implemented, we still need to update the values for *maxDelay*, *Wait* and *MAD* to evaluate in little time every possible insertion.

In San Sebastian, there are only 17 POI to POI connections that are unfeasible without transfers. For each connection requiring a transfer, we used the transfer connection with the shortest travel time (without waiting time). Then, we have calculated the period of this connection as the "least common multiple", obtaining a minimum period of 10 min and a maximum one of 30 min. Finally, we precalculated the travel times, AD, DD and DA. These calculations can become very complex in big cities with a lot of different lines running with different periods and connections involving more than one transfer. When more buses are involved in a transfer, the common period (and the number of leave times and time shifts to consider) increases significantly. This becomes an even bigger problem when, for a transfer connection, the selection of bus lines that gives the shortest travel time also depends on the leave time. The more bus lines involved in the connection, the higher the value for the least common multiple. For cities bigger than San Sebastian, a lot of memory will be required to store the precalculated values and retrieving the values can become too time consuming.

In the second approach we model transfers as direct connections. This approach is less accurate, but it is more suitable for bigger cities. We approximate the waiting time at the transfers by half of the period of the second bus of the transfer. This can be considered as an expected value of the waiting time at the transfer and it limits the approximation error to half of the period of the second bus. By using this expected waiting time, any transfer can be modelled as a direct connection with the bus travel time equal to the sum of both bus travel times, the actual transfer time and half of the period of the second bus. This approach assumes that users will always follow the same path, the one minimising the total travel time between two POIs. Spiess and Florian [31] and DeCea and Fernandez [30] presented more advanced approaches that can also be applied to model transfers as direct connections. They considered that, instead of taking always the same line, users make use of a set of attractive lines, boarding the first arriving line from the attractive set. All these approaches also imply that the proposed trip will require a repair procedure, similar to the one used in the average travel time approach.

6. Customisation

Tourist preferences vary according to a great variety of factors (nationality, age range, hobbies, etc.). Regarding public transportation, some tourists prefer not to walk more than a certain distance to reach a public transportation stop, others walk faster or slower, etc. Although adjusting these parameters to individual tourists is a research out of the scope of this paper, we have included some configuration parameters to simulate different tourist behaviours.

Some variables control the maximum and minimum values that make a path between POIs more appealing than another. These parameters can be easily adjusted to tourist's preferences.

- walkingSpeed: Although tourists walk at different speed, the algorithm has been initialised with a default walking speed of 15 min/km or 4 km/h.
- maxWalkingDistance: This parameter controls the maximum walking time between POIs in order to be reachable on foot. The default value is 30 min, equivalent to a distance of 2 km at the default speed.
- minWalkingPublicTransport: This parameter controls the minimum walking time between POIs in order to consider if they can be reached by public transportation. The default value is 7.5 min, equivalent to a distance of 500 m at the default speed.
- maxLineChanges: The maximum allowed number of line changes between two POIs can be adjusted when the average travel time is calculated. The default value is 3.
- maxDistancePoiStop: The distance within stops are searched near a POI is limited by this parameter. The default value is 1 km.

7. Experimental results

7.1. TD-TOPTW test instances

We have tested the algorithm on a set of test instances based on San Sebastian. San Sebastian is located in the Basque Country, just 20 km away from France. San Sebastian's picturesque coastline makes it a popular beach resort, being one of the most relevant tourist places of the North of Spain. Most tourists visit the city by combining public transportation with short walks. Public transportation offers a good bus service with a fixed timetable through a dense network (see Fig. 9). The local organisation in charge of the bus service has provided access to real transportation data. The network includes 467 bus stops, 26 lines and more than 65 000 direct bus connections between stops.

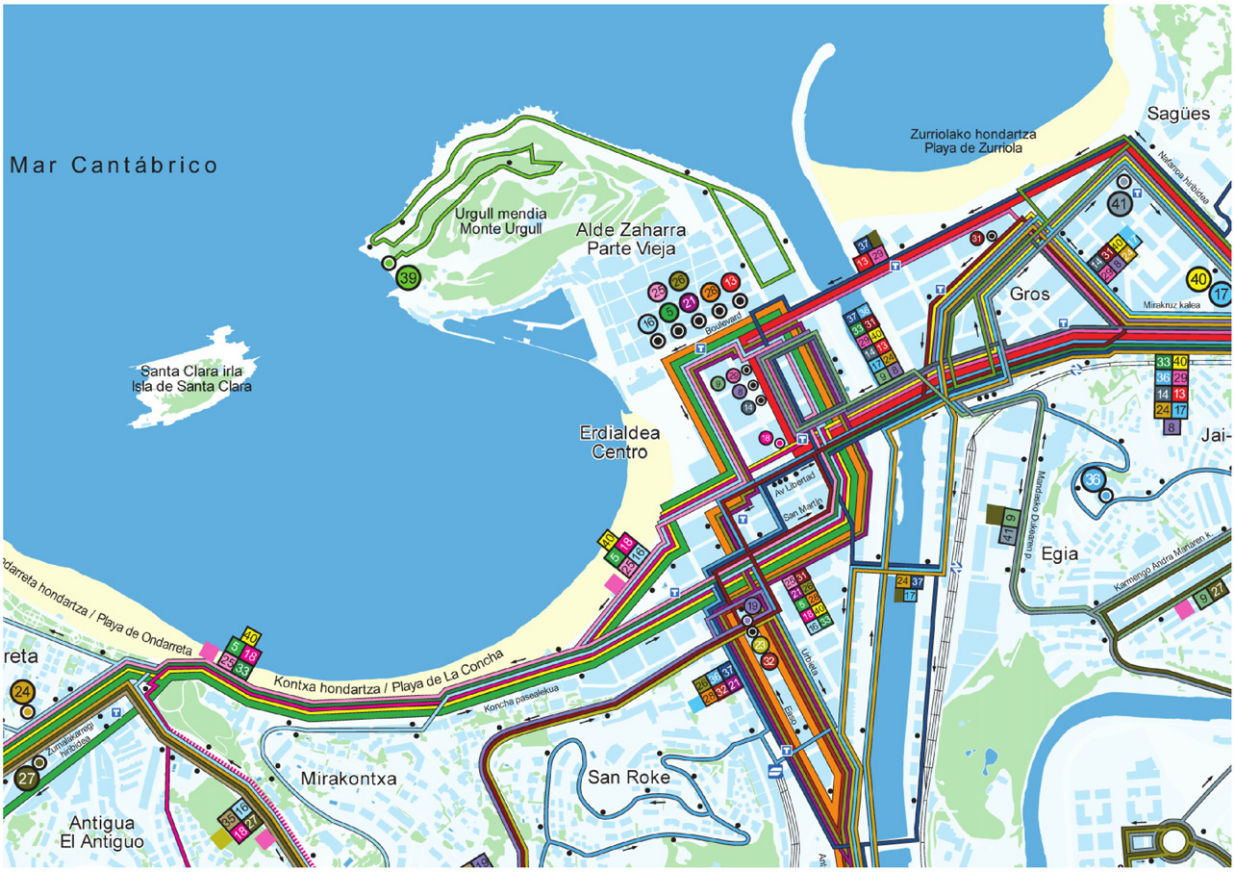

Fig. 9. Public bus network of San Sebastian.

From the tourist point of view, San Sebastian offers around 50 POIs. Although most of them are located near the three beautiful beaches and the city centre, the size of the city makes not desirable visiting all POIs on foot. We have assigned to each POI a score between 0 and 100 directly related to its tourist value. In reality, these scores will be calculated based on the tourist profile. Our optimisation goal is to obtain trips that maximise the score. Fig. 10 shows the distribution of the POIs through the city.

In order to test the strength of both approaches, we have created 28 test instances varying three different criteria:

- Number of days (routes) of the trip: We have created trips with 1 and 2 days.
- Starting POI of each day: We have chosen seven different starting POIs spread over the city.
- Length of each day: All trips start at 10 a.m. and we have established two values for the maximum length: 4 and 8 h.

Fig. 11 shows the situation of the seven different starting POIs of the trip. They are distributed through the city and they correspond to a car park (1), the main bus station (2), the train stations (3 and 4) and different hotels (5, 6 and 7).

Complete data about the setup of the test (POIs, stops, lines, etc.) is available upon request from the authors. We have carried out all computations on a personal computer Intel Core 2 Quad with 2.40 GHz processors and 2 GB Ram. The algorithm is coded using Java 1.6.

7.2. Results

Tables 4 and 5 summarise the results of the tests. The first group of columns includes the identification of the start POI and the length of each day in hours. The second group of columns gives the results for the algorithm with average travel times, including the score and the number of POIs visited during the route. We have only counted the number of visited POIs, thus the starting and ending POIs are not included in this value. The next group shows the same data for three different approaches with real travel times, including the average gap with the score of the average approach (=(score AVG−score Real)/score AVG). A negative average gap indicates that better results are obtained with the real approach. The three real approaches are real approach with no transfers (RealNoT); real approach precalculating AD, DD and DA (RealP); and real approach with transfers modelled as direct connections (RealDC).

According to the results, the trips created with all approaches are very similar. The average gap between them is below 2.7% for 1 day routes and below 2% for 2 days routes. The average travel time approach obtains slightly worse results and needs equivalent computation time. This is not surprising, since this approach only adds a repair procedure to the fast TOPTW algorithm. Thanks to the

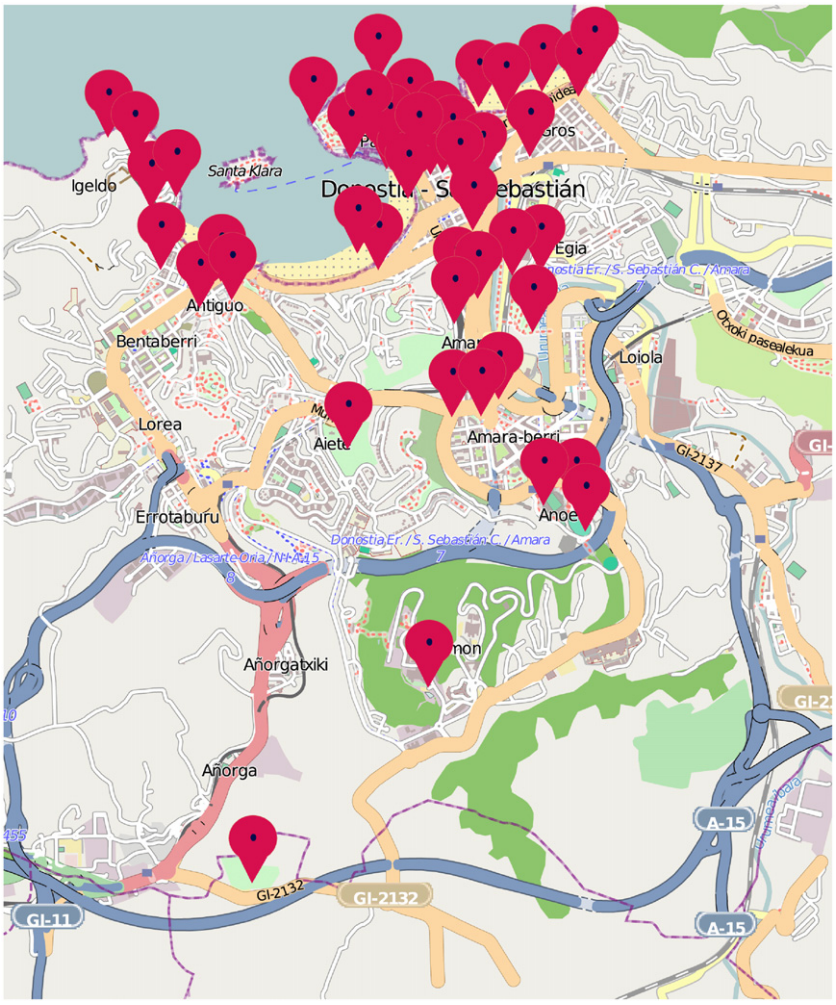

Fig. 10. POIs of San Sebastian.

Fig. 11. Distribution of starting POIs.

Table 4
Summary of results for 1 day routes.

startId	Length (h)	AVG		RealNoT		RealP		RealDC	
		Score	#	Score	#	Score	#	Score	#
1	4	1035	14	1030	14	1110	15	1095	15
	8	1745	23	1730	23	1810	24	1800	24
2	4	1070	14	1065	14	1120	15	1145	15
	8	1795	24	1780	24	1810	24	1850	25
3	4	1115	15	1145	15	1195	16	1115	15
	8	1715	23	1730	23	1860	25	1800	24
4	4	1145	15	1135	15	1180	16	1180	16
	8	1800	24	1800	24	1810	24	1860	25
5	4	1195	16	1195	16	1195	16	1195	16
	8	1790	24	1810	24	1860	25	1860	25
6	4	1035	14	1085	15	1115	15	1100	15
	8	1770	24	1795	24	1830	25	1830	25
7	4	1115	15	1115	15	1145	15	1145	15
	8	1775	24	1800	24	1825	24	1800	24
		Average gap with AVG		−0.6%		−4.0%		−3.4%	

high frequency of the public transportation in San Sebastian, the average travel times are always good approximations of the actual travel times. This partly explains why these good results are obtained. It should be noted that the complete precalculation step, in order to determine the average travel time matrix, takes around 90 min. The resulting average travel times are available from the authors upon request.

All the real calculation approaches, even with the no transfers restriction, also obtain very good results. The results confirm that not allowing any transfers is not a hard restriction, for a city with the characteristics of San Sebastian. This approach is very efficient, having very low computation times. Indeed, the computation times have the same order of magnitude as those obtained by Vansteenwegen et al. [10] solving regular TOPTW problems. Both real approaches with transfers improve the results obtained without transfers (average

Table 5
Summary of results for 2 days routes.

startId	Length (h)	AVG		RealNoT		RealP		RealDC	
		Score	#	Score	#	Score	#	Score	#
1	4	1585	21	1650	22	1655	22	1690	23
	8	2525	34	2550	34	2280	30	2585	35
2	4	1650	22	1700	23	1720	23	1770	24
	8	2610	35	2610	35	2610	35	2610	35
3	4	1645	22	1720	23	1770	24	1680	23
	8	2585	35	2610	35	2625	35	2585	35
4	4	1730	23	1730	23	1745	23	1730	23
	8	2625	35	2620	35	2610	35	2625	35
5	4	1745	23	1780	24	1810	24	1810	24
	8	2610	35	2475	33	2625	35	2610	35
6	4	1625	22	1635	22	1700	23	1720	23
	8	2550	34	2255	30	2610	35	2610	35
7	4	1710	23	1650	22	1730	23	1730	23
	8	2610	35	2610	35	2610	35	2610	35
		Average gap with AVG		0.3%		−1.5%		−2.2%	

improvement of 2.7%), although no transfers are included in the final solutions. Moreover, both real approaches with transfers obtain very similar results (average difference below 0.2%). With transfers, all movements between POIs are feasible, and thus more movements are evaluated for insertion during the search for the best solution. It was interesting to notice that transfers that appeared during the solution process were always replaced by including an extra visit near the transfer station. These results have been obtained for the city of San Sebastian, for bigger cities we expect the improvement to be larger.

For the number of POIs of San Sebastian, all the approaches are fast enough to calculate trips in real-time and are valid for creating personalised tourist trips. Based on the current calculation times, it can be expected that instances with higher number of POIs can also be solved in real-time.

Regarding the number of days (routes) of the trips, all the approaches are able to create routes of 2 or more days in a short computation time. Routes of 2 days and 8 h per day collect around 80% of the total score. This resembles the typical tourist stay at San Sebastian (a weekend), which gives enough time to visit most of the POIs. We have performed tests with up to 7 days, but with trips of 4 days, all the POIs are visited. The sum of the duration of the visits to all POIs, ignoring travel times, takes around 20 h. Thus it is not possible to visit all of them in 2 days (16 h) and probably even in 3 days. Although the algorithm is able to solve 7 day trips in real-time for San Sebastian, from the tourist point of view so long trips are very uncommon in a city like San Sebastian.

A detailed analysis of the results confirms the benefits of integrating public transportation in the trip planning. Due to the proximity of most POIs, public transportation is mainly used to arrive or leave from the city centre, and also to reach distant POIs. For example, Fig. 12 shows a 4 h route starting from POI 1. POI 1, a car park far from the city centre, has not been included in this figure to increase the readability of the example, its location is shown in Fig. 11. In this route, three buses are taken (paths with public transportation have been represented by broken lines). The first one leaves from the car park to a POI that is not in the city centre. The second one departs from this POI to the city centre. And the final bus returns back from the last POI visited in the city centre to the car park. The remaining travel time within the route is walking time. In total, this route has around 70 min of walking time, 20 min of bus time and 8 min of waiting time for the bus.

On the other hand, on a 4 h route starting from POI 5 (located in the city centre), no bus is taken because all the visited POIs are within walking distance. An 8 h and 4 days route starting from POI 3 and visiting all the POIs, has around 6 h of total walking time, 90 min of bus time, 40 min of waiting time and takes 10 buses.

8. Conclusions and future work

This paper shows how the use of public transportation can be integrated in personalised tourist trips by solving the time-dependent team orienteering problem with time windows (TD-TOPTW). We proposed and tested different approaches to include public transportation in planning trips.

The first approach is based on a precalculation step, where we calculate the average travel times between all pairs of POIs. With these average travel times, we solve the TD-TOPTW as a regular TOPTW. In the resulting trip proposal, we will have to adapt the arrival and leave times of the visits according to the differences between the average travel times and the (time dependent) real travel times. As a consequence, it might be necessary to remove one or more visits that have become unfeasible, before proposing the trip to a tourist.

The second approach is based on a fast evaluation of the possible insertion of an extra POI. By introducing a few new concepts, it becomes possible to evaluate each insertion locally and efficiently. We present three variants of this approach. The first one only considers direct public transportation connections, without transfers. The second variant is based on precalculating all required values for the first variant, and also for the connections that require a transfer. The third variant consists of modelling transfers as direct connections.

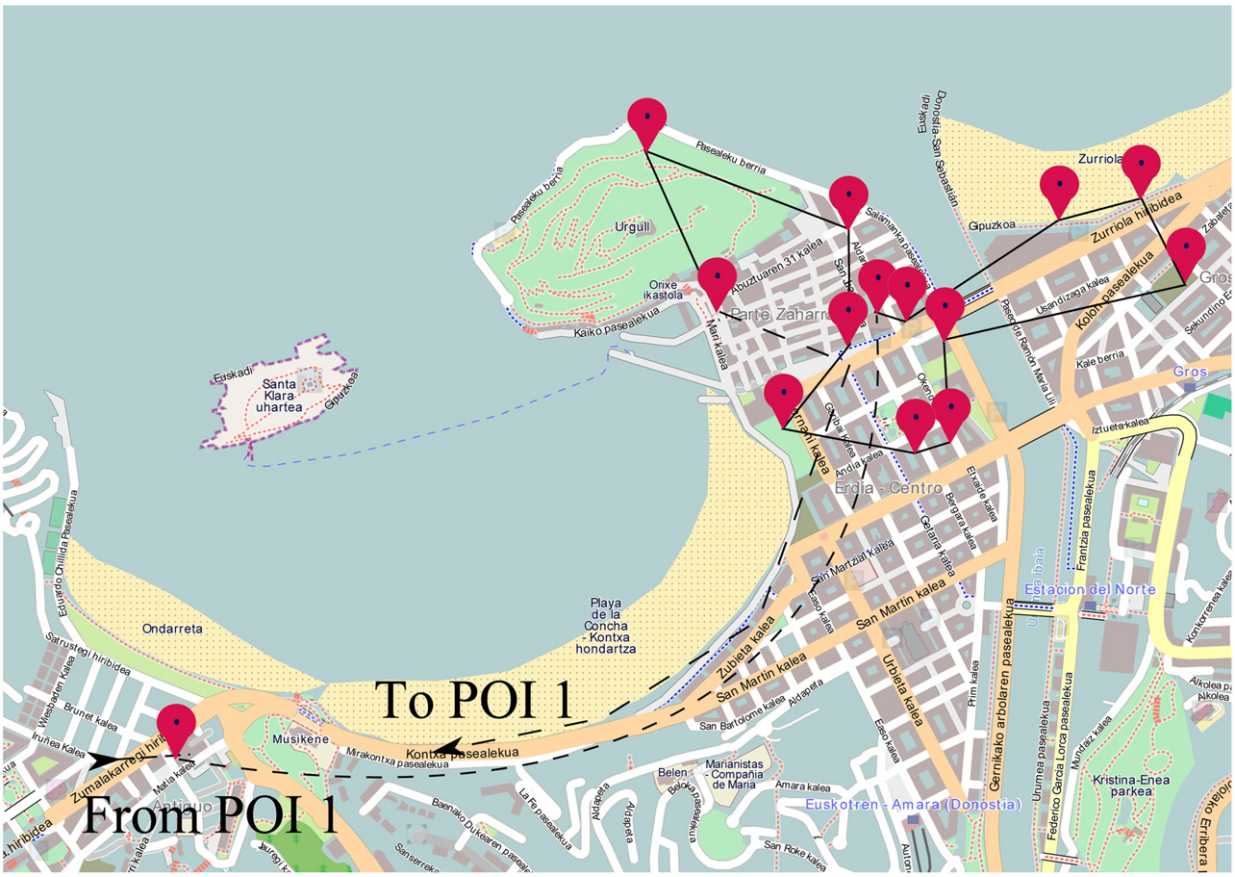

Fig. 12. Example route.

We have implemented and tested all the approaches for a set of test instances based on San Sebastian, a medium size city with around 50 POIs, 26 public transportation lines and 467 stops. All approaches succeeded in calculating personalised trips of 1 and 2 days in real-time.

The real travel time approaches outperform the approach with average travel times. The calculation times are comparable but the obtained results are slightly better. The best results are obtained with the real travel time approach when transfers are considered. Considering transfers widens the search space and leads to better results, although no transfers are present in the final solutions. For bigger cities, considering transfers is even more important and the improvement introduced by real-time approaches with transfers is expected to be higher.

Both real-time approaches with transfers obtain very similar results for the city of San Sebastian. Modelling transfers as direct connections in big cities requires to apply advanced models (such as the ones presented by Spiess and Florian [31] or De Cea and Fernandez [30]). With these models, although we would no longer work with real travel and waiting times, we would keep the simplicity and efficiency of the real approach with no transfers.

The real-time approach based on the precalculation works with real travel and waiting times. However, it could be difficult to apply in big cities where different lines with different periods and more than one transfer are involved in the connections. Apart from the difficulties of the precalculations, if high "least common multiple" periods are required, the memory requirements to store and read AD, DD and DA values could break the real-time requirement of the algorithm.

For the personalised electronic tourist guide (PET) and other tourist applications, these algorithms provide a new quality feature. One of the key functionalities that tourists demand when moving within an unknown area is public transportation information. Being able to create trips in real-time that take public transportation into account is a big step for future PET development.

The next step is to implement the algorithm in a real mobile device, in order to test the perceived tourist quality of the routes with the help of the staff from the local tourist organisation and with real tourists visiting a city. Regarding the algorithm, we plan to acquire real data about other tourist cities with different public transportation network topologies and POI distribution, to test the algorithm in a different environment. For bigger cities, an optimised version of the time-dependent shortest path calculation should be implemented for the average travel time approach.

Acknowledgements

The authors would like to thank the Basque Government for partially funding this work through the neurebide and etourgune projects and to the Centre for Industrial Management of the Katholieke Universiteit Leuven for hosting Ander Garcia as a guest researcher during 2009. Finally, authors would like to thank Javier Vallejo from "Compañia del Tranvia de San Sebastian" for providing real data about the public transportation network of the city.

References

[1] Brown B, Chalmers M. Tourism and mobile technology. In: ECSCW'03: Proceedings of the eighth conference on European conference on computer supported cooperative work. Norwell, MA, USA: Kluwer Academic Publishers; 2003. p. 335–54.
[2] Beer T, Fuchs M, Höpken W, Werthner H, Rasinger J. Caips: a context-aware information push service in tourism. In: Information and communication technologies in tourism 2007. Ljubljana, Slovenia: SpringerWien; 2007. p. 129–49.
[3] Schmidt-Belz B, Laamanen H, Poslad S, Zipf A. Location-based mobile tourist services-first user experiences. In: Information and communication technologies in tourism 2003. Ljubljana, Slovenia: SpringerWienNewYork; 2003. p. 115–23.
[4] Stroobants R. Mobile tourist guides. Tech. Rep., Katholieke Universiteit Leuven, Belgium; 2006.
[5] Souffriau W, Vansteenwegen P, Vertommen J, Vanden Berghe G, Van Oudheusden D. A personalized tourist trip design algorithm for mobile tourist guides. Applied Artificial Intelligence 2008;22(10):964–85.
[6] Garcia A, Linaza M, Arbelaitz O, Vansteenwegen P. Intelligent routing system for a personalised electronic tourist guide. In: Information and communication technologies in tourism 2009. Amsterdam, The Netherlands: SpringerWienNewYork; 2009. p. 185–97.
[7] Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D. Metaheuristics for tourist trip planning. In: Geiger M, Habenicht W, Sevaux M, Sorensen K, editors. Metaheuristics in the service industry (Lecture notes in economics and mathematical systems)Springer Verlag; 2009. p. 15–31.
[8] Tsiligirides T. Heuristic methods applied to orienteering. Journal of the Operational Research Society 1984;35(9):797–809.
[9] Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D. The city trip planner: a tourist expert system. Expert Systems with Applications 2011;38:6540–6.
[10] Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D. Iterated local search for the team orienteering problem with time windows. Computers & Operations Research 2009;36(12):3281–90.
[11] Vansteenwegen P. Planning in tourism and public transportation. PhD thesis, Centre for Industrial Management, Katholieke Universiteit Leuven, Belgium; 2008.
[12] Kramer R, Modsching M, ten Hagen K, Gretzel U. Behavioural impacts of mobile tour guides. In: Information and communication technologies in tourism 2007. Ljubljana, Slovenia: SpringerVienna; 2007. p. 109–18.
[13] Maruyama A, Shibata N, Murata Y, Yasumoto K, Ito M. A personal tourism navigation system to support traveling multiple destinations with time restrictions. International Conference on Advanced Information Networking and Applications 2004;2:18.
[14] Zenker B, Ludwig B, Schrader J. Rose: assisting pedestrians to find preferred events and comfortable public transport connections. In: Mobility '09: proceedings of the 6th international conference on mobile technology, application & systems. ACM; 2009. p. 1–5.
[15] Vansteenwegen P, Souffriau W, Van Oudheusden D. The orienteering problem: a survey. European Journal of Operational Research 2011;209(1):1–10.
[16] Savelsbergh MWP. Local search in routing problems with time windows. Annals of Operations Research 1985;4(1):285–305.
[17] Righini G, Salani M. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers & Operations Research 2009;36(4):1191–203.
[18] Tricoire F, Romauch M, Doerner K, Hartl R. Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 2010;37(2):351–67.
[19] Montemanni R, Gambardella L. Ant colony system for team orienteering problems with time windows. Foundations of Computing and Decision Sciences 2009;34(4):287–306.
[20] Fomin F, Lingas A. Approximation algorithms for time-dependent orienteering. Information Processing Letters 2002;83(2):57–62.
[21] Pyrga E, Schulz F, Wagner D, Zaroliagis C. Efficient models for timetable information in public transportation systems. Journal of Experimental Algorithmics 2008;12:1–39.
[22] Ding B, Yu J, Qin L. Finding time-dependent shortest paths over large graphs. In: EDBT '08: proceedings of the 11th international conference on extending database technology. New York, NY, USA: ACM; 2008. p. 205–16.
[23] Zografos KG, Androutsopoulos KN. Algorithms for itinerary planning in multimodal transportation networks. IEEE Transactions on Intelligent Transportation Systems 2008;9(1):175–84.
[24] Chiu DKW, Lee OKF, Leung H-F, Au EWK, Wong MCW. A multi-modal agent based mobile route advisory system for public transport network. In: Proceedings of the 38th annual Hawaii international conference on System sciences, 2005 (HICSS '05); 2005. p. 92.2.
[25] Tumas G, Ricci F. Personalized mobile city transport advisory system. In: Information and communication technologies in tourism 2009. Amsterdam, The Netherlands: SpringerWienNewYork; 2009. p. 173–83.
[26] Lourenço HR, Martin O, Stutzle T. Iterated local search. In: Kochenberger FGG, editor. Handbook of metaheuristicsKluwer Academic Publishers; 2003. p. 321–53.
[27] Brodal GS, Jacob R. Time-dependent networks as models to achieve fast exact time-table queries. In: 3rd workshop on algorithmic methods and models for optimization of railways (ATMOS 2003). Electronic notes in theoretical computer science (92). Department of computer science, University of Aarhus. Demmark: Elsevier; 2004.
[28] Delling D, Sanders P, Schultes D, Wagner D. Engineering route planning algorithms, vol. 2. Berlin, Heidelberg: Springer-Verlag; 2009. p. 117–39.
[29] Bauer R, Delling D, Wagner D. Experimental study of speed up techniques for timetable information systems. Networks 2011;57(1):38–52.
[30] De Cea J, Fernandez E. Transit assignment to minimal routes. An efficient new algorithm. Traffic Engineering & Control 1989;30(10):491–4.
[31] Spiess H, Florian M. Optimal strategies: a new assignment model for transit networks. Transportation Research Part B: Methodological 1989;23(2):83–102.