Expert Systems with Applications 42 (2015) 7303–7316

Contents lists available at ScienceDirect

Expert Systems with Applications

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

The eCOMPASS multimodal tourist tour planner

Damianos Gavalas [a,f,*], Vlasios Kasapakis [a,f], Charalampos Konstantopoulos [b,f], Grammati Pantziou [c,f], Nikolaos Vathis [d,f], Christos Zaroliagis [e,f]

[a] Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Greece
[b] Department of Informatics, University of Piraeus, Piraeus, Greece
[c] Department of Informatics, Technological Educational Institution of Athens, Athens, Greece
[d] School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
[e] Department of Computer Engineering & Informatics, University of Patras, Patras, Greece
[f] Computer Technology Institute and Press 'Diophantus' (CTI), Patras, Greece

ARTICLE INFO

Article history:
Available online 1 June 2015

Keywords:
Tourist Trip Design Problem
Multimodal tour planning
Urban transportation
Orienteering Problem
Time window
Time dependent travel time
Context awareness
Web service
Mobile application

ABSTRACT

Tour planning represents a challenging task for individuals visiting unfamiliar tourist destinations, mainly due to the availability of numerous attractions (points of interest, POIs) and the complexity of metropolitan public transit networks. Several web and mobile tourist city guides already support personalized tour recommendations. However, they exclusively consider walking tours; namely, they fail in motivating tourists to use public transportation for reaching far located important POIs, thereby compromising the perceived overall attractiveness of recommended tours. In this paper, we introduce eCOMPASS, a context-aware web/mobile application which derives personalized multimodal tours via selected urban attractions. eCOMPASS is the only available research or commercial tour planner that assists the way arounds of tourists through public transit. Far beyond than just providing navigational aid, eCOMPASS incorporates multimodality (i.e. time dependency) within its routing logic aiming at deriving near-optimal sequencing of POIs along recommended tours so as to best utilize the time available for sightseeing and minimize waiting time at transit stops. Further advancing the state of the art, eCOMPASS allows users to define arbitrary start/end locations (e.g. the current location of a mobile user) rather than choosing among a fixed set of locations. Last, eCOMPASS may assist in scheduling lunch breaks at affordable restaurants, conveniently located along the recommended tours. The provision of the above mentioned unique features of eCOMPASS is based on modeling and solving a complex optimization problem which takes into account a long list of problem variables and constraints. This paper describes the routing algorithm which comprises the core functionality of eCOMPASS. Further, it discusses the implementation details of the web and mobile eCOMPASS applications using the metropolitan areas of Athens (Greece) and Berlin (Germany) as case studies. Evaluation results report positive user attitude as to the tour planning output with respect to attractiveness, meaningfulness and the overall perceived utility.

1. Introduction

Tourists visiting urban destinations typically deal with the challenge of making a feasible plan in order to visit the most interesting attractions (points of interest, POIs) in their available time span. The filtering of most important POIs (amongst the many available) and their time-sequencing along the planned tourist itineraries is a particularly laborious task requiring skilled interaction with a multitude of online resources (Brown & Chalmers, 2003; Souffriau & Vansteenwegen, 2010).

The situation is further complicated when considering the complexity of metropolitan transit networks commonly used by tourists to move from a POI to another, whenever walking is not an option, due to distance constraints. Tourists are typically unfamiliar with and intimidated by the nuances of the public transit systems in their destination areas, thereby making transit transfers a cumbersome exercise. Tourists are especially reluctant in using bus networks as they feel they do not have the acquired local knowledge to negotiate them efficiently, while also running the risk of leaving the tourism space and entering a terra incognita,

* Corresponding author at: Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Greece.
E-mail addresses: dgavalas@aegean.gr (D. Gavalas), v.kasapakis@aegean.gr (V. Kasapakis), konstant@unipi.gr (C. Konstantopoulos), pantziou@teiath.gr (G. Pantziou), nvathis@softlab.ntua.gr (N. Vathis), zaro@ceid.upatras.gr (C. Zaroliagis).

http://dx.doi.org/10.1016/j.eswa.2015.05.046
0957-4174/

should they use a wrong service or take a wrong direction (Lew & McKercher, 2006). An interesting aspect highlighted by field studies is that tourists seek to maximize time spent at a place by minimizing transit time. Tourists typically opt for public transportation when pedestrian walking is long enough to challenge their strength and endurance. Even then, any delays incurred on stops (waiting to board on the next transiting service) are highly undesirable, given the limited time budget spent on the tourist destination (Lew & McKercher, 2006).

The above discussion underlines the need for ICT tools to assist the way arounds of tourist transfers among POIs, either walking or using public transit. Such tools typically appear in the form of personalized tourist guides (PETs) which tackle a problem commonly termed as Tourist Trip Design Problem (TTDP) (Gavalas, Konstantopoulos, Mastakas, & Pantziou, 2014b; Schaller & Elsweiler, 2014). TTDP refers to a tour planning problem for tourists interested in visiting multiple POIs. Solving the TTDP entails deriving daily tourist tours comprising ordered sets of POIs that match tourist preferences, thereby maximizing tourist satisfaction (typically termed 'profit'), while taking into account a multitude of parameters and constraints (e.g., distances among POIs, time estimated for visiting each POI, POIs' opening hours) and respecting the time available for sightseeing on daily basis.

TTDP has been a subject of intensive research in the recent years. As a result, the city tour planning tools (essentially, TTDP solvers) have proliferated, incorporating an array of useful services (Borràs, Moreno, & Valls, 2014). However, existing tools still miss several practical aspects, hence, compromising their utility in realistic tourist scenarios. Firstly, they exclusively consider walking as the only option provided for tourist transfers. This is certainly impractical when considering POIs scattered throughout large metropolitan areas or when tourists lodge in hotels far from main attraction areas. Secondly, tours generated by existing tools exclusively comprise visits to attractions, allocating no time for lunch/rest breaks. Lunch recommendations would probably be much appreciated by users overwhelmed by the – too many – restaurant options typically offered in touristic regions. Thirdly, available tools allow users to select the start/end points of their itineraries among a fixed set of lodging and/or landmark locations. This is clearly restrictive as mobile users would likely be reluctant to move to a specified location only to faithfully follow a recommended tour. Practical scenarios could involve users requesting tours starting/ending at arbitrary locations (the starting point would, most probably, be their location at query time).

Herein, we present *eCOMPASS*, a context-aware web/mobile application which addresses the above described shortcomings of existing approaches to TTDP deriving personalized multimodal daily tourist itineraries. A focal design objective of eCOMPASS has been to support tourists moving around either walking or using public transit represents. Transit timetables are considered so as to derive travel itineraries which involve the fastest transfer option among walking and transit, taking into account delays incurred on transit stops. Certainly, visits to POIs are ordered so as to make best use of transit services; namely to minimize delays on transit stops and be able to accommodate more POI visits thereby maximizing collected profit. Further, eCOMPASS caters for scheduling lunch breaks through recommending affordable restaurants conveniently located along the tourist tour. Last, our proposed tool allows users to define – different – arbitrary start/end locations for their daily tours.

The core engine of eCOMPASS is based on a novel cluster-based heuristic approach, the *SlackRoutes*. The main incentive behind our approach is to motivate visits to topology (i.e. touristic) areas featuring high density of 'good' (i.e. highly profitable) candidate vertices, even if those are located relatively far. SlackRoutes takes into account time dependency (i.e. multimodality) in calculating travel times from one vertex (i.e. POI) to another. The aim is to derive high quality routes (i.e. maximizing the total collected profit) and minimize the time delays incurred in transit stops, while not sacrificing the time efficiency required for online applications. A preliminary version of SlackRoutes appears in Gavalas, Konstantopoulos, Mastakas, Pantziou, and Vathis (2014).

The remainder of this article is structured as follows: Section 2 reviews relevant approaches both in the algorithmic domain and the tourist tour planning software tools. Section 3 presents the SlackRoutes tour planning algorithm. Section 4 overviews the architecture and discusses the system implementation details of eCOMPASS. Section 5 presents user evaluation results, while Section 6 concludes our work and suggests directions for future research.

2. Related Work

TTDP has received considerable attention in the recent years with several – mainly heuristic – algorithmic methods proposed to solve it (Gavalas et al., 2014b). Those methods approach the problem from different angles, resulting in diverse problem models, which consider different problem variables and constraints. A consolidated listing of input parameters considered in proposed TTDP models follows:

- A set of candidate POIs, each associated with a number of attributes (e.g. type, location, opening days/hours, entrance fee, etc).
- The number of tours to be generated, based upon the period of stay of the user at the tourist destination.
- The 'profit' of each POI, denoting its relevant importance.
- The anticipated visit duration of an average user at a POI.
- The 24×7 time-dependent travel times among POIs, i.e. tourists are assumed to use all modes of transport available at the tourist destination, including public transportation, walking, car, bicycle, etc.
- The daily time budget B that a tourist wishes to spend on visiting sights; the overall daily tour duration (i.e. the sum of visiting times plus the overall time spent for moving from a POI to another) should be kept below B.

The objective in TTDP modeling is to derive a set of near-optimal daily, disjoint itineraries (ordered visits to POIs), each comprising a subset of available (candidate) POIs so as to maximize tourist satisfaction (i.e. the overall collected profit); the derived tours should respect user constraints/POI attributes and satisfy the daily time budget available for sightseeing. Acknowledging that tourists have different attitudes or perceptions on the same attractions (Han, Guan, & Duan, 2014), several problem parameters (e.g. the profit and the estimated visit duration of POIs) may be adjusted according to user preferences.

The baseline combinatorial optimization problem typically used for modeling TTDP is the Orienteering Problem (OP) (Vansteenwegen, Souffriau, & Van Oudheusden, 2011). In the OP, given a starting node s, a terminal node t and a positive time limit (budget) B, the goal is to find a path from s to t (or tour if $s \equiv t$) with length at most B such that the total profit of the visited nodes is maximized. Clearly, the OP may be used to model the simplest version of the TTDP wherein the goal is to find a single tour that maximizes the profit collected from visited POIs, within a given time budget (time allowed for sightseeing in a single day).

One of the earliest OP-based tour planning approaches (De Choudhury et al., 2010) considered tours along POIs mined from geo-tagged photos of Flickr; the photos' timestamps have been used to estimate the visiting time at each POI as well as the transit times among places. Using a variant of the OP, Gionis,

Lappas, Pelechrinis, and Terzi (2014) categorized POIs such that recommended tours are constrained by a POI category visit order (e.g., first visit a museum, then a park and then a beach). A similar approach has been followed by Bolzoni, Helmer, Wellenzohn, Gamper, and Andritsos (2014) who addressed the max-*n* type constraint (Gavalas, Konstantopoulos, Mastakas, & Pantziou, 2014a), i.e., limiting the number of visits at POIs of certain categories. Likewise, a modified OP modeling has been proposed by Lim (2015) to constrain tour planning by a mandatory POI category, which corresponds to the POI category a user is mostly interested in.

Brilhante, Macedo, Nardini, Perego, and Renso (2015) presented TripBuilder,[1] a web tool which employs a two-step process upon POI collections retrieved from the Wikipedia and albums of geo-referenced photos from Flickr. Firstly, the step of deriving an optimal set of POIs (based on POI popularity and user preferences) is modeled as an instance of the Generalized Maximum Coverage problem (Cohen & Katzir, 2008). Then, the selected POIs are sequenced along a sightseeing itinerary by a heuristic algorithm addressing an instance of the Traveling Salesman Problem. Kurata and Hara (2014) presented CT-Planner4,[2] a web-based tourist tour relying on a genetic algorithm which solves the Selective Traveling Salesman Problem (STSP) (Laporte & Martello, 1990). The solver starts with *n* random initial tours and ends up to a tour plan with the highest utility score (i.e. profit) through iteratively performing a crossover/mutation procedure. The algorithm's parameters are tuned so that the computation completes within a second.

Notably, all the aforementioned approaches only consider the case of single tour planning and assume POIs available to visit on a 24×7 basis. Several extensions of the OP have been successfully applied to model more complex TTDP versions. Among them, the Team Orienteering Problem with Time Windows (TOPTW) (Hu & Lim, 2014; Labadi, Melechovský, & Wolfler Calvo, 2011; Vansteenwegen, Souffriau, Vanden Berghe, & Van Oudheusden, 2009) has been the most frequently studied. TOPTW extends OP to multiple tours (hence, enabling planning across several daily tours) and considers visits to locations within a predefined time window (this allows modeling opening and closing hours of POIs). Among the alternative TOPTW algorithms, the iterated local search (ILS) algorithm (Vansteenwegen et al., 2009) is considered most suitable for real-time TTDP applications as it derives routes of reasonable quality (on average, less than 5% gap from the best known solution) fairly fast (less than 7 s for up to 200 POIs and $m = 4$ daily tours). ILS (combined with a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic proposed by Feo and Resende (1989)) is known to comprise the core tour planning engine of CityTripPlanner[3] (Vansteenwegen, Souffriau, Vanden Berghe, & Van Oudheusden, 2011), a commercial web/mobile city tour planner. The CityTripPlanner users are allowed to edit derived tours, remove unwanted POIs and/or adjust visiting time scheduled for particular POIs (in all cases, edits trigger recalculation of tours). The start/end locations may be selected among a fixed set of hotels (i.e. user's lodging) and landmarks.

Garcia, Vansteenwegen, Arbelaitz, Souffriau, and Linaza (2013) proposed algorithmic solutions for the time-dependent TOPTW (TDTOPTW). TDTOPTW considers time dependency in the estimation of the time required to move from one location to another and, therefore, it is suitable for modeling multi-modal transports among POIs. The authors presented two different approaches to solve the TDTOPTW. The first approach involves a pre-calculation step, computing the average travel times between all pairs of POIs, allowing reducing the TDTOPTW to a regular TOPTW, solved using the ILS. The second approach uses time-dependent travel times but it based on the assumption of periodic service schedules; this assumption, clearly, does not hold in realistic urban transportation networks, wherein arrival/departure frequencies typically vary within the services operational periods while deviations from planned service time schedules commonly occur due to non-predictable events.

Time-dependency has also been taken into account by Hasuike, Katagiri, Tsubaki, and Tsuda (2014) for calculating traveling times among POIs; yet, under a different problem setting, wherein the set of POIs to be visited are predetermined. The problem has been formulated as a time expanded network and an exact algorithm based on dynamic programming has been proposed to derive appropriate sightseeing routes and time scheduling of visits. However, the authors admit that their approach is not applicable to large scale networks, hence, it is not suitable for real-time TTDP applications when considering realistic tourist destination instances.

In this article, we introduce the eCOMPASS tourist tour planner, which advances the current state-of-the-art with respect to several aspects:

- eCOMPASS takes into account time-dependent travel times among POIs; namely, in addition to walking tours, it considers the option of using public transit for moving around, making no assumption on periodic service schedules. Note that this is far more than just providing navigational aid (i.e. transit route instructions) for moving from one location to another. Instead, eCOMPASS incorporates multimodality within its routing logic (the time needed to move from a POI to another depends on the departure time and the utilized transportation mode) aiming at deriving near-optimal sequencing of POIs along recommended tours so as to best utilize the time available for sightseeing and minimize waiting time at transit stops.
- Users typically seek stop overs at affordable and conveniently located restaurants along their tours. Existing tour planning approaches overlook this requirement and derive itineraries exclusively comprising visits to attractions. eCOMPASS allows users to schedule lunch breaks through recommending restaurants based on both their price range and their location so as not to require long detours away from attraction areas.
- Existing research and commercial prototypes restrict users to selecting the start/end points of their daily itineraries among a fixed set of locations (typically a list of accommodations and/or landmarks). This restriction is dictated by the fact that the travel times among all possible location pairs should be computed offline to ensure fast response to user queries. eCOMPASS follows a novel approach that overcomes this restriction and allows users to define arbitrary start/end locations (e.g. their current location).

3. The SlackRoutes tour planning algorithm

SlackRoutes is a TDTOPTW solver which comprises the algorithmic core of eCOMPASS. The algorithm is given a complete directed graph $G = (V,E)$ where V denotes the set of locations with $N = |V|$, a set $P = \{p_1, p_2, \ldots p_{|P|}\} \subseteq V$, denoting the set of POIs and derives m routes (one for each day of staying at the destination) comprising an ordered set of POIs in P; each route is bounded by a time budget B. The starting and terminal locations of the rth route are denoted as s_r and t_r, respectively. Accordingly, st_r and et_r denote the starting, ending time, respectively of the rth route, $r = 1, 2, \ldots m$.

The main attributes of each node $p_i \in P$ are: the service or visiting time (v_i), the profit gained by visiting p_i (profit_i), and the time window for each day $TW_{ir} = [\text{open}_{ir}, \text{close}_{ir}]$; $r = 1, 2, \ldots m$ (a POI may have different time windows per day).

[1] http://tripbuilder.isti.cnr.it/.
[2] http://ctplanner.jp/ctp4/index-e.html.
[3] http://www.citytripplanner.com/.

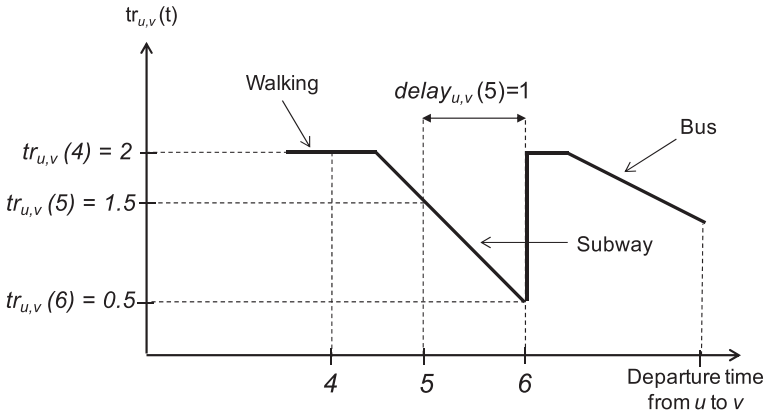


Fig. 1. Traveling time from u to v as a function of the departure time from u (travel time profile of $u \rightarrow v$).

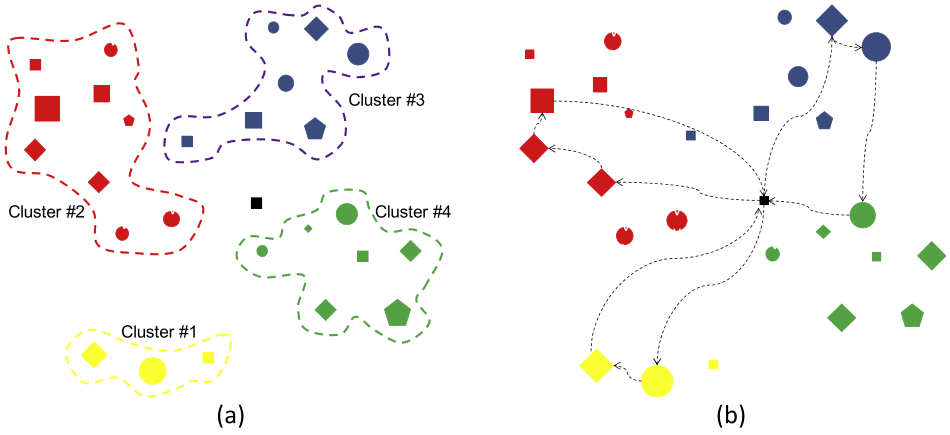


Fig. 2. Illustration of SlackRoutes execution phases: (a) initial topology (geometric shapes indicate POI categories, while their size denote POIs profit values) clustered based on geographic criteria (preprocessing phase); (b) adjustment of POI profit values based on user profile (i.e. user preferences upon POI categories indicated through user queries) and example output (tours) derived by SlackRoutes.

The travel time $tr_{u,v}$ among locations u and v is time-dependent since a user may either walk or move through a non-periodic transit service (we assume that the fastest option is preferable). Essentially $tr_{u,v}(t)$, termed 'multimodal travel time profile' is a piecewise linear function: $tr_{u,v}(t) = \min\{walking_{u,v}, delay_{u,v}(t) + travtime_{u,v}(t)\}$, where $walking_{u,v}$ denotes the time for walking from u to v (constant), $delay_{u,v}(t)$ is the time spent on a transit stop (when leaving from u to v at time t) waiting for the next service to arrive and $travtime_{u,v}(t)$ is the total travel time spent after the initial boarding at u till arriving at v. This time may include any delay spent at intermediate stops or time spent for walking between two stops along the route. For our purposes, we require to know pairwise fastest routes between POIs for all departure times of the day. Namely, for each pair of POIs we compute all the non-dominated pairs $(dep_{u,v}, tr_{u,v})$ where $dep_{u,v}(t)$ is a departure time and $tr_{u,v}(t)$ is the corresponding travel time from u to v. We assume that a pair (dep, tr) dominates a pair (dep', tr') iff $dep > d'dep' + tr' \leqslant dep + tr$.

Fig. 1 illustrates the travel time profile for transfers from node u to node v. When a user departs from u to v at time 4, it is preferable to walk since walking will take 2 time units ($tr_{u,v}(4) = 2$) while the next transit (subway) service departs at 6 and takes 0.5 time units of traveling time to arrive at v (namely, the user is expected to arrive at 6 when walking and at 6.5 when waiting for the subway). Subway becomes the preferable option when leaving u between 4.5 and 6 time units.

Furthermore, for each POI p_i in a route r, SlackRoutes utilizes the following variables:

$wait_i$	The waiting time at p_i before its time window starts; $wait_i = \max\{0, open_i - arrive_i\}$.
$start_i$	The starting time of the visit at p_i; $start_i = arrive_i + wait_i$.
$leave_i$	The departure time from p_i; $leave_i = start_i + visit_i$.
$arrive_i$	The arrival time at p_i; $arrive_i = leave_{prev(i)} + tr_{prev(i),l}(leave_{prev(i)})$ where prev(i) is the previous node of p_i in route r. We assume that $arrive_{s_r} = st_r$.
$maxStart_i$	The latest time the visit at p_i can start without violating the time windows of the nodes following p_i; $maxStart_i = min\{close_{ir}, \max\{t : t + tr_{i,next(i)}(t) \leqslant maxStart_{next(i)}\} - visit_i\}$, where next($i$) is the node following p_i in r. We assume that $maxStart_{t_r} = et_r$.
$slack_i$	How long the arrival at p_i may be delayed without affecting the feasibility of following visits; $slack_i = maxStart_i - arrive_i$. Note that if the value of $slack_i$ is close to 0, it is highly unlikely the insertion of a new POI between $p_{prev(i)}$ and p_i to be feasible.

Note that the variables *wait*, *start* and *arrive* have been defined in (Vansteenwegen et al., 2009). The *leave*, *maxStart* and *slack* variables are introduced in this work.

3.1. Execution phases of SlackRoutes

SlackRoutes comprises an offline (preprocessing) and an online phase (executed upon the receipt of user queries); the latter involves three main execution routines. A pseudocode implementation of SlackRoutes is shown in Algorithm 1.

Algorithm 1: *SlackRoutes*

```
1.  Preprocessing phase: Cluster the POIs and construct the listOfClusters
2.  while listOfClusters is not empty do
3.      ClusterSet ← listOfClusters.pop                    } Route initialization
4.      RouteInit (ClusterSet)                             }
5       iterations ← 0
6.      while iterations < maxIterations do                } Route creation
7.          Repeat                                         }
8.              Insert                                     }
9.          until no further insertion is feasible         }
10.         if currentSolution is the best found then      }
11.             bestFoundSolution ← currentSolution        }
12.             iterations ← 0                             }
13.         end if                                         }
14.         Shake                                          }
15          Iterations ← Iterations + 1                    }
16.     end while                                          }
17.     remove all POIs visited in the currentSolution
18. end while
```

3.1.1. Preprocessing phase

The preprocessing phase contributes to the efficient handling of user queries. It takes as input an initial topology, i.e. a set of POIs belonging to disjoint categories/sets (i.e. monuments, museums, churches, squares, etc); in the example topology of Fig. 2(a), geometric shapes indicate POI categories, while their size denote POIs profit values. The initial topology is partitioned into a number of clusters (listOfClusters) based on geographical criteria, using the global *k*-means algorithm (Likas, Vlassis, & Verbeek, 2003).

The main incentives behind the clustering process are: (a) to force successive visits to POIs grouped in the same cluster in the online phase; this, firstly, reduces the solution space saving execution time and, secondly, motivates walking instead of transit transfers (POIs grouped within the same cluster are likely to be within walking distance); (b) to indicate areas featuring high density of 'promising' (i.e. highly profitable) POIs.

3.1.2. Online execution phase

The online execution phase is triggered upon the receipt of user queries with preferences upon specific POI categories (e.g. preference to visit archaeological sites rather than modern art museums). Those preferences are used to adjust the profit values of POIs (note that in Fig. 2(b), the profit of rhombus and circular POIs is increased while the profit of square and polygon POIs is reduced, compared to their original values).

From a high level perspective, the online phase tries out different route initializations (lines 3–4); starting with an initial solution, SlackRoutes executes an iterated local search procedure inserting POIs along the initial routes until no further insertion is feasible (lines 7–9) and then it perturbs (shakes) derived solutions in hope of escaping local optima and achieving further improvement (line 13). The solution with the highest overall profit is returned to the user.

Routes initialization (lines 3–4). A list of disjoint listOfClusters (derived from the preprocessing phase) is considered, originally arranged in cluster quality (C_q) order. The C_q metric is inspired by the *h*-index, that is, a cluster with $C_q = h$ should include at least h POIs with profit greater or equal to h. The intuition behind the C_q ordering is to motivate visits to clusters highly dense in POIs with high profit values, even if those POIs are located relatively far. The m clusters with the highest C_q value are selected from listOfClusters. Then, one POI from each of these clusters is inserted into each of the originally empty m routes (below, we explain the criterion used to select the initial POI for each route).

Routes creation (lines 6–16). Routes creation involves an insertion step (lines 7–9), which performs consecutive POI insertions and a shake step (line 13) which attempts to improve the originally derived solutions.

When considering a route r, an **Insert** routine is iteratively executed, provided that POIs are visited within their time windows and the daily time budget in not exceeded, until no further improvement is possible. The Insert routine considers the insertion of all candidate POIs p_i after each POI p_{i_k}. [4] For each feasible position, it calculates the weight w_i^k of p_i:

$$w_i^k = profit_i \cdot A_i^k; A_i^k = \frac{\sum_{j=1}^{k} slack_{i_j} + slack_i + \sum_{j=k+1}^{n'+1} slack_{i_j}}{n'+2}$$

The quantity A_i^k denotes the average slack value among all POIs on r, after the insertion of p_i after the kth POI along r (the route will then include $n'+1$ visits). Note that the $slack_{i_{n'+1}}$ corresponds to the slack of the final leg of the trip between the last POI $p_{i_{n'}}$ and the final destination t_r. Notice also that a large value of A_i^k implies that after the insertion of p_i, there will be many possibilities left for inserting new POIs along each leg of the trip (that is, prior and after visiting p_i). Eventually, the POI with the highest weight w_i, where $w_i = \max(w_i^k)$, is inserted into the position k. Notably, unlike most existing iterated local search procedures, e.g. the ILS algorithm (Vansteenwegen et al., 2009), SlackRoutes involves a global rather than a local decision perspective regarding possible insertion positions as it considers the effect of POIs insertion along the whole route.

SlackRoutes does not allow a route to visit a cluster more than once: a POI insertion can take place before or after a POI of its containing cluster, if such a POI exists, otherwise the insertion can take place only between POIs of different clusters. This restriction saves execution time by restricting the solution space and reduces the transit transfers, since transfers between POIs of the same cluster are typically done by walking.

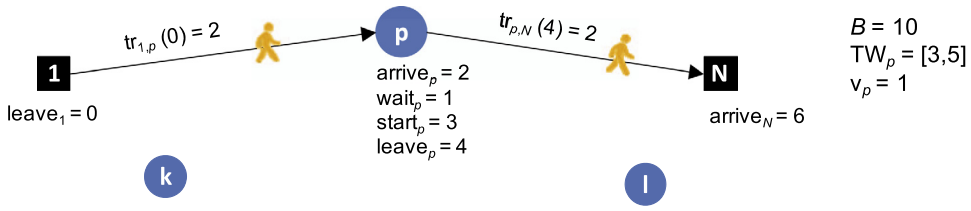
The **Shake** routine comprises a solution perturbation step, wherein a number of consecutive POIs are removed from each route; the insertion procedure is then executed attempting to escape local optima. An example output derived by SlackRoutes is illustrated in Fig. 2(b).

3.2. Example execution of SlackRoutes

Fig. 3 illustrates the details of a feasible route, i.e. a route that conforms to the time windows of inserted nodes and the total available time budget. The route starts from location *1*, ends at location *N* and involves a visit to POI *p*, 2 time units later on (the traveling time is $tr_{1,p}(0) = 2$). The visit at *p* can take place between 3 and 5 (TW_p = [3,5]), hence it starts after waiting for 1 time unit at *p* ($wait_p$ = 1). The visit at *p* lasts for 1 time unit (v_p = 1). The tourist then leaves from *p* and arrives at the end location *N*, 6 time units after the beginning of the route ($tr_{p,N}(4) = 2$, as shown in Fig. 1, for $p \equiv u$ and $N \equiv v$), hence, leaving 4 time units of spare time (B = 10 time units). The maxStart values are calculated starting from the end location and iterating backwards along the route: $\text{maxStart}_N = 10 = B$, $\text{maxStart}_p = 5$ (since $\text{close}_p = 5$), etc. The slack variable values are calculated accordingly, e.g. $\text{slack}_N = \text{maxStart}_N - \text{arrive}_N = 10 - 6 = 4$, etc.

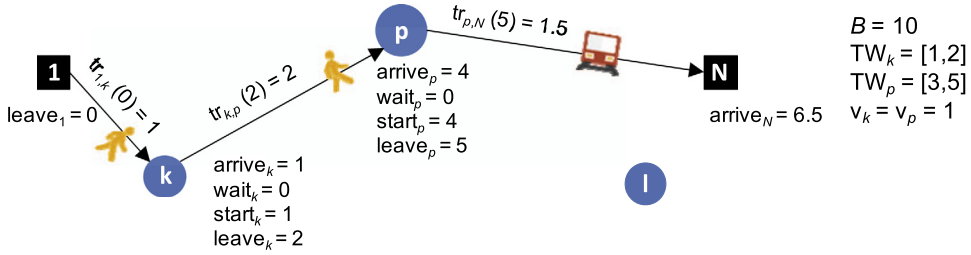
Two unused nodes (POIs), *k* and *l*, with the same profit are subsequently examined as candidates for the next insertion along the route. Fig. 4 examines the option of inserting *k* before *p* (inserting *k*

[4] If $k = 0$, the POI *i* is placed first of all POIs of the route.



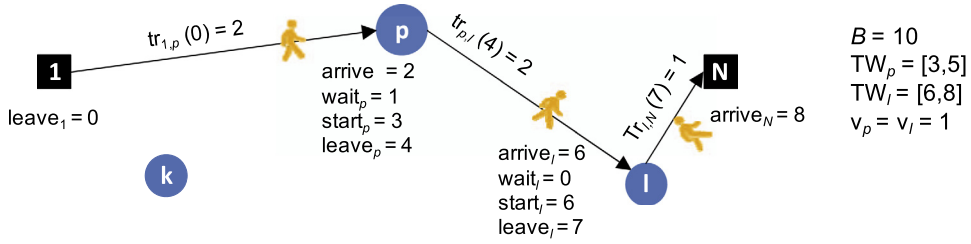
	$arrive_i$	$wait_i$	$start_i$	$leave_i$	$tr_{i,next(i)}(leave_i)$	$maxStart_i$	$slack_i$
1	-	-	-	0	2	3	3
p	2	1	3	4	2	5	3
N	6	-	-	-	-	10	4

Fig. 3. (a) The initial route 1-*p*-*N*.



	$arrive_i$	$wait_i$	$start_i$	$leave_i$	$tr_{i,next(i)}(leave_i)$	$maxStart_i$	$slack_i$
1	-	-	-	0	1	1	1
k	1	0	1	2	2	2	1
p	4	0	4	5	1.5	5	1
N	6.5	-	-	-	-	10	3.5

Fig. 4. Details of the route after inserting POI *k*.



	$arrive_i$	$wait_i$	$start_i$	$leave_i$	$tr_{i,next(i)}(leave_i)$	$maxStart_i$	$slack_i$
1	-	-	-	0	2	3	3
p	2	1	3	4	2	5	3
l	6	0	6	7	1	8	2
N	8	-	-	-	-	10	2

Fig. 5. Details of the route after inserting POI *l*.

after *p* would be infeasible due to the time window of *k*, $TW_k = [1,2]$). Note that $tr_{p,N}(5) \neq tr_{p,N}(4)$: when leaving from *p* at 4, it takes shorter time to walk (2 time units), while when leaving from *p* at 5, it is preferable to use the subway (one time unit waiting at the subway station and another half unit for the actual transfer). Based on the tabular data of Fig. 4, the insertion of *k* before *p* would have average slack equal to $A_k = \frac{1+1+1+3.5}{4} = \frac{6.5}{4}$.

Fig. 5 examines the option of inserting *l* after *p* (*l* could not be inserted prior to *p* due to its time window ($TW_l = [6,8]$) which cannot be served prior to visiting *p* ($TW_p = [3,5]$). Based on the calculated maxStart/slack variable values, the average slack associated with this insertion will be $A_l = \frac{3+3+2+2}{4} = \frac{10}{4} > A_k$. Having larger average slack and identical profit value, the insertion of *l* is more preferable than that of *k* (that is, the insertion of *l* leaves more room for additional visits along the route), hence *l* is inserted in the route (after *p*).

3.3. Incorporating lunch breaks

Tourists commonly chose restaurants on the basis of hard/soft constraints, preferences (e.g. price range, cuisine, etc.) and their location as they prefer not to considerably deviate from their

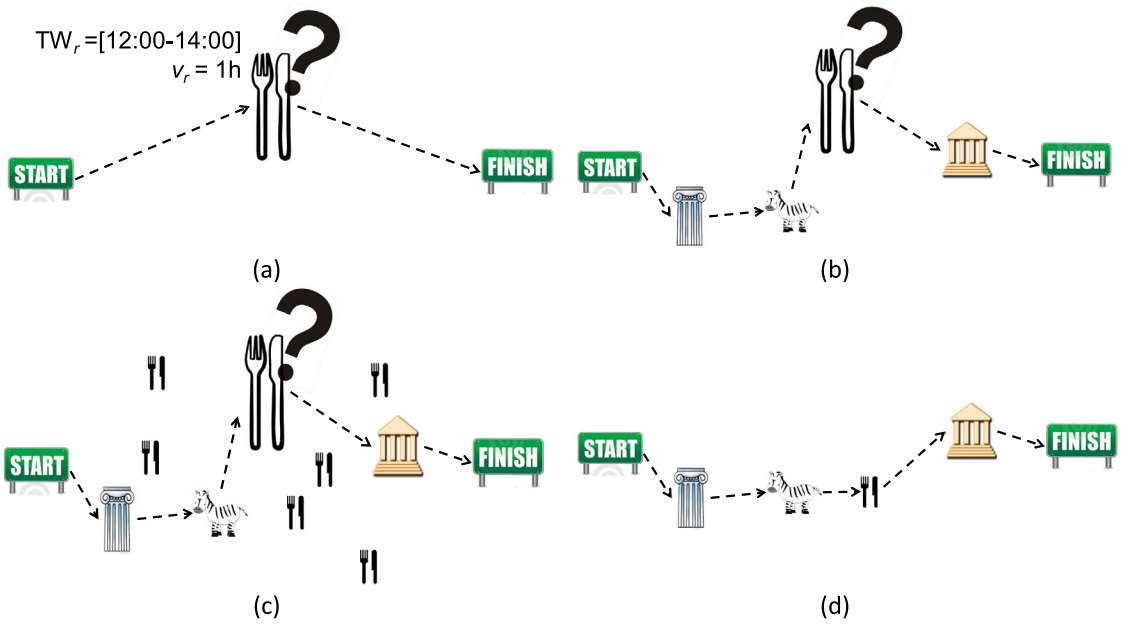


Fig. 6. Process of incorporating lunch breaks.

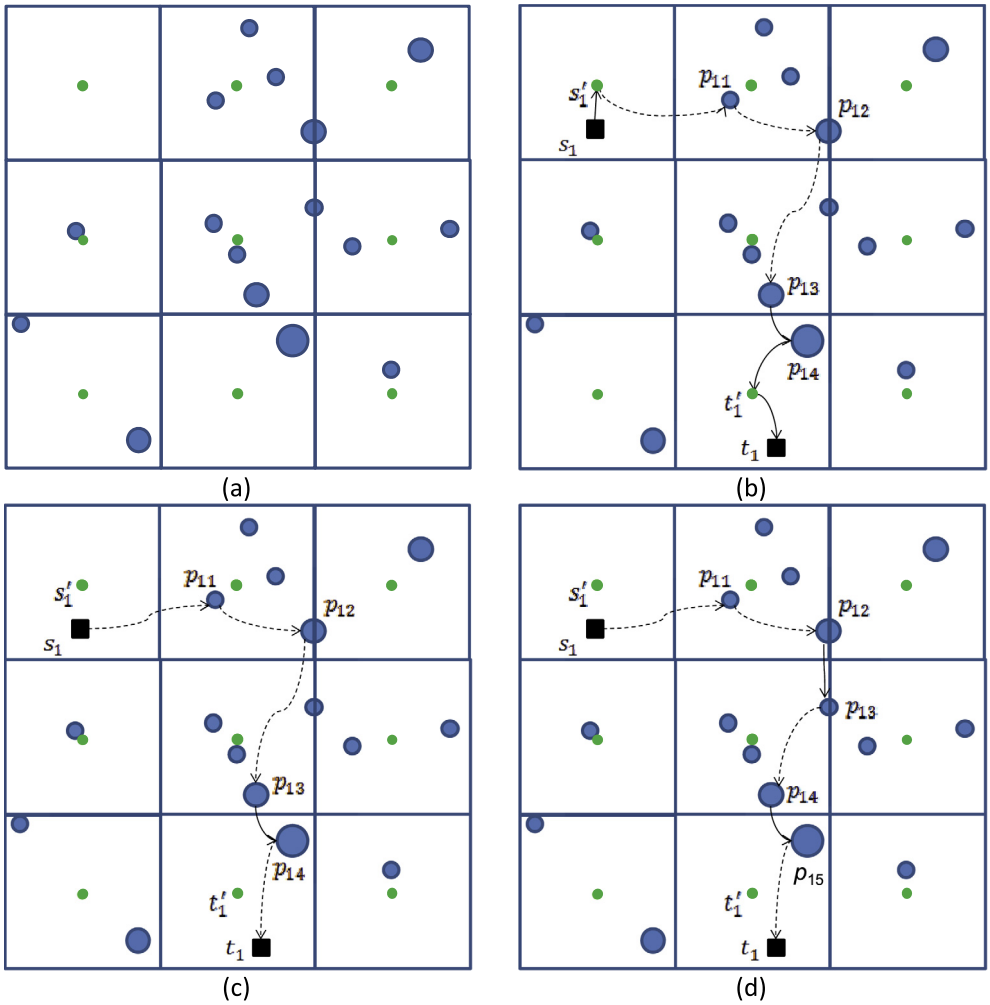


Fig. 7. Consideration of arbitrary start/end locations: (a) Preprocessing phase; (b)–(d) online phase.

Table 1
Compiled locations, transit network data, memory & execution time requirements for the Athens and Berlin instances.

		Athens	Berlin
Locations	Regions	144 (=12 × 12) km^2	256 (=16 × 16) km^2
	POIs	113	185
	Hotels	100	105
	Restaurants	100	103
	Cafes	100	84
Clusters		11	18
Transit network (GTFS) data	Transportation authority	Athens Urban Transport Organization	Berliner Verkehrsbetriebe (U-Bahn, Tram, Bus)/ Deutsche Bahn (S-Bahn)
	Transit services	3 subway lines, 3 tram lines and 287 bus lines	9 lines U-Bahn, 15 lines S-Bahn, 22 tram lines and 147 bus lines
	Transit network stops	7825	3213
Memory size for storing travel profiles (GB)		3.33	5.76
Average query execution time (ms) for time budget 10:00–18:00	1 tour	350	400
	2 tours	600	650
	3 tours	750	1000
	4 tours	1100	1650

sightseeing routes. Given the numerous restaurant options, the selection of a suitable restaurant may be even more cumbersome than scheduling POI visits. However, existing tourist tour planners exclusively consider visits to POIs ignoring the need for lunch/rest breaks.

eCOMPASS addresses this issue scheduling lunch breaks in affordable restaurants located near the tourist tour. From a tour planning point of view, restaurants may be considered as a separate set of POIs with identical profit; among these POIs it is compulsory to visit only one.

The route creation process starts with inserting a 'dummy' node (Fig. 6(a)) with its time window and visiting time properties provided by the user (the user indicates her/his preferred time span and duration of lunch break). Then, SlackRoutes executes as normal, inserting visits to POIs prior/after the visit to the restaurant (Fig. 6(b)). It is noted that the travel cost between the dummy node and a POI is set equal to the shortest time dependent travel cost between the POI and a restaurant. Next, all available restaurants within the specified budget are considered (Fig. 6(c)). The one requiring the shortest time to reach is chosen to visit at the same position held by the dummy node (Fig. 6(d)). In the case that the derived route is infeasible, POIs of the route are iteratively removed (in profit ascending order) until route feasibility is attained.

3.4. Support for arbitrary start/end itinerary locations

(TD)TOPTW modeling involves an asymmetric distance matrix storing travel times among all nodes. The distance matrix should be precomputed to ensure fast response to user queries. The offline calculation becomes more crucial when considering time dependent travel times which are far more costly to compute. Due to the above considerations, available tourist tour planners restrict choices for itineraries start/end locations among a fixed set of selected hotels and/or landmarks (those locations are included in the set of locations V). This restriction, though, counters the reasonable expectation to define arbitrary start/end route locations (e.g. the current location of the user), which will only be known at query time. In the sequel we present an algorithm that addresses this issue.

The preprocessing phase described in Section 3.1 is extended so as to further perform a partitioning of the tourist area into small square regions (e.g. 500 m × 500 m), covering the whole geographical area where POIs are located in. Within each region R_i a central location c_{R_i} is chosen. For instance, in Fig. 7(a), the area is partitioned in nine areas, where the green dots denote the centers. Consider the complete directed graph $G = (V,E)$, where V consists of all the POIs and centers. Then for each pair of locations (i,j) in

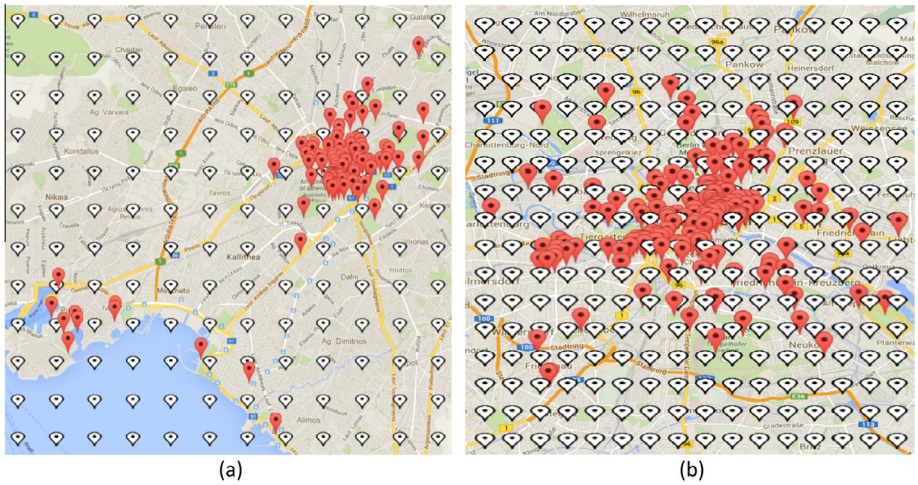


Fig. 8. Region centers (white markers) and POIs (red markers) for (a) Athens, Greece; (b) Berlin, Germany. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

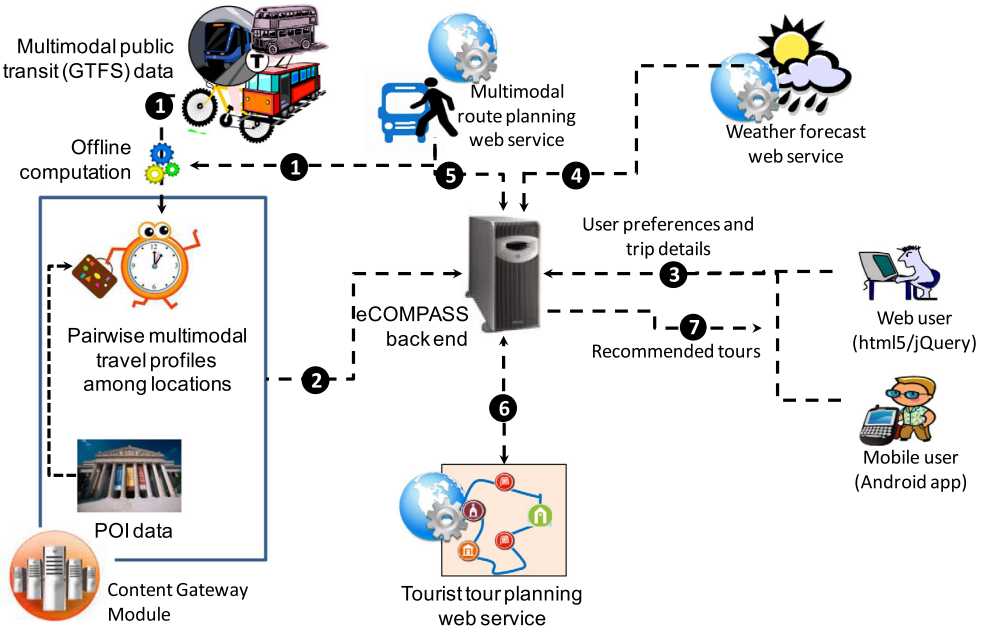


Fig. 9. eCOMPASS system overview.

V, the travel profile is calculated. Next, the online phase of the algorithm proceeds as follows:

- For each route r_i, $i = 1,..,m$, (i) find the centers s_i' and t_i' of the regions where the arbitrary end points s_i and t_i belong to. (ii) Compute the walking travel times t_{s_i} and t_{t_i} between (s_i, s_i') and (t_i, t_i'), respectively. Note that due to the small size of the regions, it is fast to compute the walking distances, while walking is probably faster than public transportation. (iii) Fix the start/end time of the route to be the arrival times at s_i' and t_i' ($st_i + t_{s_i}$ and $et_i - t_{t_i}$, respectively).
- Run the SlackRoutes and connect (s_i, s_i') and (t_i, t_i') as first/last route legs (Fig. 7(b), where solid and dashed lines denote walking and transit transfers, respectively).
- Restore the original start/end locations and times and connect s_i and t_i with the first and last POI in the designed route, respectively (Fig. 7(c)).
- Repair the route inserting new POIs, if possible (Fig. 7(d)).

4. eCOMPASS system implementation

This Section focuses on the implementation details of the eCOMPASS tour planner prototype.

4.1. Content database

We have compiled two location collections from the urban areas of Athens (Greece) and Berlin (Germany); each collection comprises region centers, attractions (POIs), hotels, restaurants and cafes (see Table 1).

POIs are classified in the following categories: museums & art galleries, nature (parks, lakes), archaeological sites, neighborhoods & squares (pedestrian streets, markets, squares, scenic/historic walking areas), churches & religious heritage, monuments & landmarks (palaces, historical monuments, sculptures), marines & ports. The metadata stored for each POI include: title; geo-coordinates (latitude, longitude); category; profit; visiting time; opening/closing hours for each week day; indication of whether it is 'open air' (i.e. unsuitable to visit in rainy or hot days) or not; entrance fee; indication of accessibility facilities; short description; photograph(s); address; telephone; official website URL; Wikipedia entry URL; average user rating, overall number of uploaded user ratings.

The POIs have been compiled from various tourist portals[5] and web services offering open APIs.[6] Profits have been set in a 1–100 scale and visiting times vary from 5 min (e.g. for some outdoor statues) to 2 h (e.g. for some not-miss museums and wide-area archaeological sites). About half of the POIs are outdoors and always visitable (24 h time windows) while the remainder are associated with relatively wide, largely overlapped time windows (typically around 8 h). POIs are grouped in $\frac{N}{10}$ disjoint clusters. Most of selected hotels, restaurants and cafes are situated around main attraction areas.

4.2. User profile & trip data

The eCOMPASS client applications provide user-friendly dialogs to allow the user designate the trip details/constraints as well as his/her user profile (trip preferences). In particular, trip details include the number of days to be spent at the destination, the arrival date, the start/end time and location for daily trips.

Moreover, the user rates his/her preferences on POI categories (in 0–10 scale). The originally set POI profits and visiting times are then adjusted accordingly to match the user profile. The adjustment of profit values also takes into account the average rating score ur_{p_i} received by other mobile users for each POI p_i as to the average rating score avg_{ur} among all POIs:

$$p_i = p_i + \frac{p_i \cdot (rating_{cat(p_i)} - 5)}{10} + \frac{p_i \cdot (ur_{p_i} - avg_{ur}) \cdot \min\{1, r_{p_i}/10\}}{avg_{ur}}, \quad = 1 \ldots N$$

where, $rating_{cat(p_i)}$ denotes the rating assigned by the user to the category which POI p_i belongs to. The factor $\min\{1, r_{p_i}/10\}$ ensures that ratings provided by others users are taken into account only when the number of ratings r_{p_i} received for p_i are more than 10 so that statistical validity is guaranteed.

[5] http://www.tripadvisor.com/, http://index.pois.gr/.
[6] https://developers.google.com/places/documentation/.

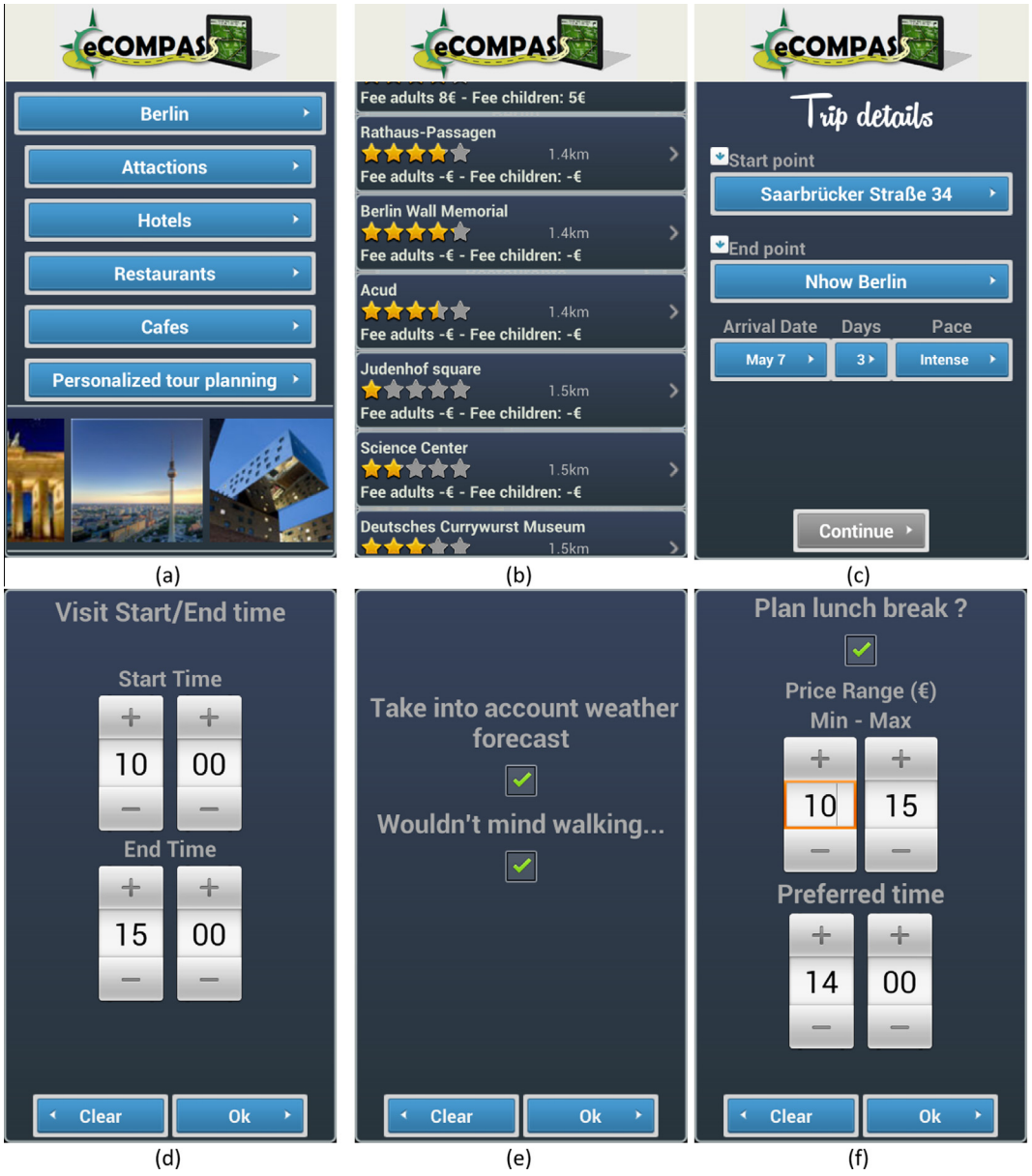


Fig. 10. Screenshots taken from the eCOMPASS mobile (Android) client application.

The user also customizes 'advanced' settings with respect to: visit pace (relaxed, medium, intense), which determines the walking speed; whether s/he wishes weather forecast to be taken into account (if yes, no visits to 'open air' attractions will be scheduled on rainy/hot days); preference on walking rather than using public transit (public transportation will be recommended only for overly long transfers); whether lunch breaks should be scheduled (if yes, the preferred time span and maximum lunch budget are also specified) (see Fig. 8).

4.3. Calculation of multimodal travel time profiles

In our implementation, we have used the GTFS[7] data of the transit networks deployed in the metropolitan areas of Athens and Berlin (see Table 1). Using the method of Dibbelt, Pajor, and Wagner (2012), we compute offline pairwise full (24 h range) multimodal time-dependent travel time profiles among all locations stored in the content database (POIs, region centers, hotels, restaurants, cafes). We also maintain the walking time among pairs of POIs, provided that this is up to 50% longer than using the transit network at any departure time within the day. Note that the current implementation does not consider bike transfers as access/egress modes to public transit. The overall travel time information is stored in a three-dimensional array of size $N \times N \times 1440$, where N is the number of specified locations/POIs and 1440 ($=24 \times 60$) the time steps/minutes per day. This memory structure ensures instant access to time-dependent travel times, given a specified pair of POIs (u, v), upon receiving a user query. The memory size needed to store travel profiles is reported in Table 1.

4.4. eCOMPASS architecture and application workflow

eCOMPASS adopts a coarse-grained service-oriented architecture (SOA) approach, wherein the business logic is composed of

[7] The General Transit Feed Specification (GTFS – https://developers.google.com/transit/gtfs/reference) defines a common format for public transportation schedules and associated geographic information (locations of stops, description of routes, timetables of individual trips and other schedule data). Such information is necessary for any multimodal route planning service to calculate the time required to travel from a location to another via public transit.

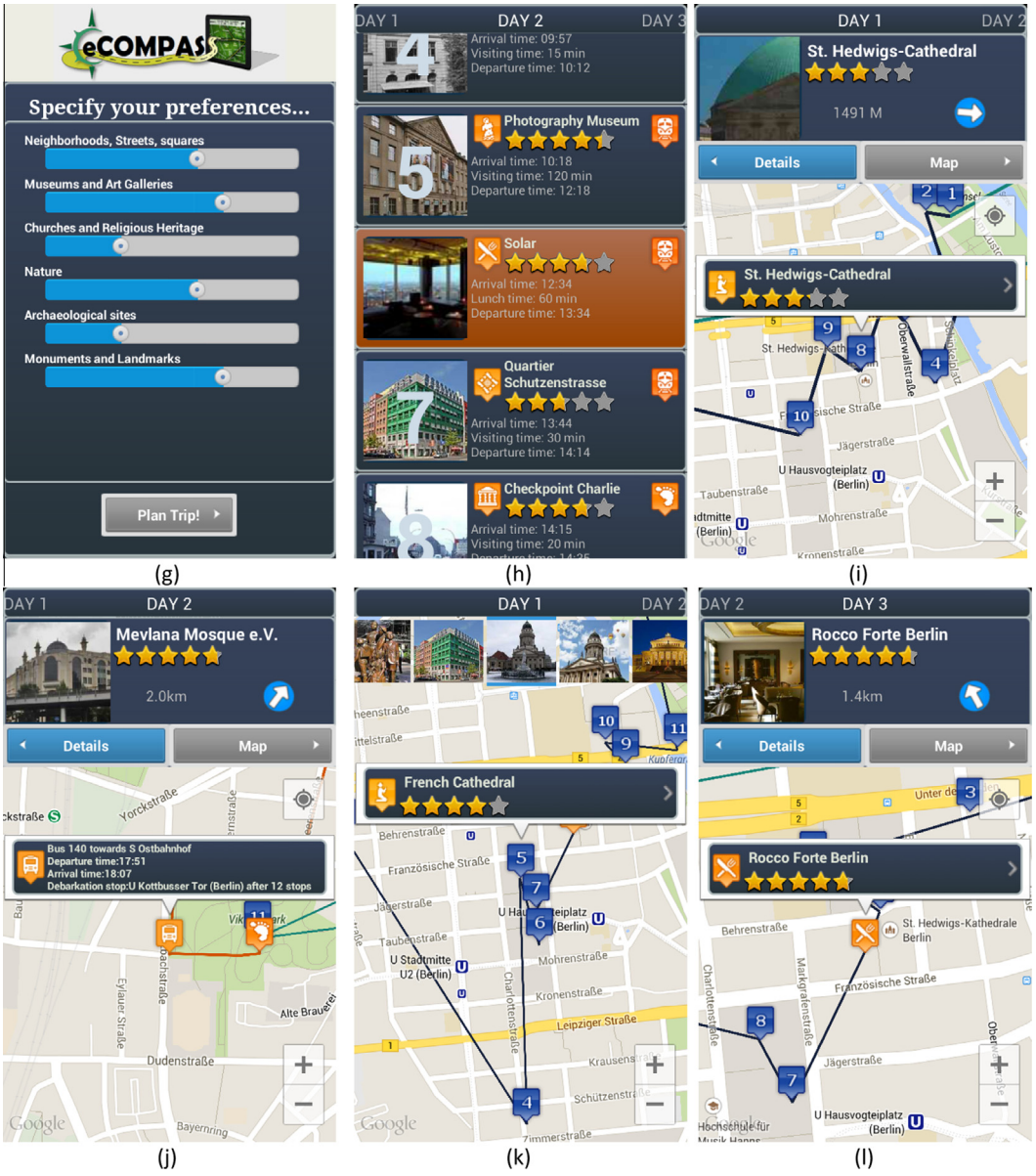


Fig. 10 (*continued*)

loosely coupled web services, combined together to deliver personalized services either through traditional web interfaces or thin mobile clients. The web client application (http://ecompass.aegean.gr/) is implemented in HTML5 and jQuery. The mobile client application (https://www.youtube.com/watch?v=BVT__mWOoso) has been developed on the Android 4.4 platform; the tour planner for Berlin may be downloaded from http://ecompass.aegean.gr/eCompass.apk.

Fig. 9 provides a high-level overview of the eCOMPASS architecture as well as the steps followed upon receiving user queries. In particular, the application workflow within eCOMPASS comprises the following phases/steps:

Offline phase:

1. The pairwise full multimodal travel time profiles among all locations stored in the content database are computed based on timetable (GTFS) data (see Section 4.3). Also, POIs are grouped in disjoint clusters based on geographical criteria.
2. POIs metadata (see Section 4.1) and travel profiles among POIs (see Section 4.3) are stored in memory structures on the server side.
3. Online phase:
4. User queries are sent along with the user profile and the trip details (see Section 4.2).
5. A weather web service (Yahoo! weather[8]) is contacted to deliver a weather forecast for the trip dates (in case that the user wishes weather information to be taken into account in the tour planning logic).
6. In case that arbitrary start/end tour locations are defined, a multimodal route planning service is contacted to estimate the walking travel time from the start/end locations to the nearest region center (see Section 3.4).
7. The tour planning web service derives the personalized daily tourist tours.

[8] https://developer.yahoo.com/weather/.

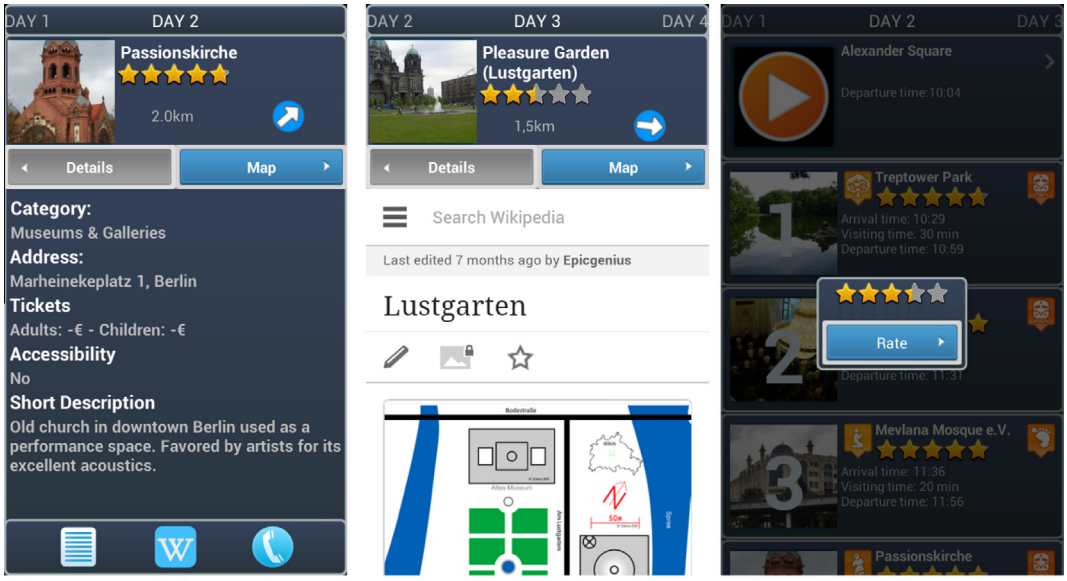


Fig. 10 (*continued*)

8. The tours are returned (in JSON format) to the requesting client application; the tour description is rendered and visualized on a map or list-based graphical interface.

Fig. 10 exhibits representative screens captured by the eCOMPASS mobile client application. The main application features are highlighted in the menu of Fig. 10(a); those include listings of attractions, hotels, restaurant and cafes (sortable by name, price, rating or distance – see Fig. 10(b)) further to the main tour planning functionality.

The next five screens present the user settings used to personalize the derived tourist tours. The user specifies the start and end location of each itinerary (see Fig. 10(c)), allowed to choose among available hotels, selected city landmarks (e.g. central squares), arbitrary locations (pointed on a map interface) and current location (yield through GPS fix). The user also indicates his/her scheduled arrival date, the number of days to be spent at the destination (this information is required to retrieve the opening hours of POIs and transit travel profiles for those days as well as the weather forecast) and the preferred walking pace (to adjust the estimated walking travel times). Optionally, the user may: modify the default start/end time for each daily itinerary (see Fig. 10(d)); indicate willingness to take weather conditions into account and preference to walk rather than taking short transit rides (see Fig. 10(e)); plan lunch breaks on restaurants of certain budget at specific time (see Fig. 10(f)). Last, the user indicates preference on separate POI categories, thereby adjusting default POI profit values accordingly, as explained in Section 4.2 (see Fig. 10(g)).

Recommended tours may be visualized in both list (see Fig. 10(h)) and map (see Fig. 10(i)) views. The list view illustrates the visiting order of recommended POIs along with their title, category, rating, estimated arrival/departure time and visiting duration. A walking/transit icon placed on each list item may be tapped to yield walk or time-dependent transit directions from the previous POI towards the current one (see Fig. 10(j)). The dual view shown in Fig. 10(k) combines a slide show (image gallery) of recommended POIs with a map view of the itinerary (upon tapping a POI's picture, the map zooms at the selected POI's location). Restaurants recommended for lunch breaks are designated with different background color and marker in the list and map view, respectively (see Fig. 10(h) and (l)).

The user may retrieve further information for selected POIs, including address, telephone, entrance fee, accessibility facilities, short description (see Fig. 10(m)). Special icons may be tapped to load the POI's official website or the respective Wikipedia page within the application space (see Fig. 10(n)). Each POI screen also shows a distance estimate among the user's current position and the POI's location as well as a compass indicator icon pointing towards the POI's location. The user may also retrieve a map with markers indicating his/her current location and the POI's location. Last, the user may upload ratings for visited POIs (see Fig. 10(o)); those ratings are fed back to the POIs database, eventually affecting the profit values assigned to POIs (see Section 4.2).

The users may remove any POI from the proposed tours; removed POIs may be restored later on. Furthermore, the start/end point and time of a specific daily itinerary may be edited without affecting the start/end points/times for the remaining itineraries. On the incident of modifying any user setting, the tour planning web service is reinvoked and the itineraries are recalculated. The last recommended daily tours are stored locally to enable fast reloading and avoid unnecessary server invocations.

4.5. Web services implementation

The web services utilized within the eCOMPASS system (i.e. the tourist tour planning web service and the multimodal route planning web service) are based on the RESTful architectural style. A front-end and a back-end part have been implemented for each service. The front-end is written in Ruby using the Sinatra framework. It accepts user input in the form of a URI and is responsible for sanitizing the input before passing it to the back-end. Then, it dispatches the input to the correct back-end process, and parses its output. Finally, it converts the output to JSON format. The back-end is written in C++ and is highly optimized for both online and offline speed, so as to minimize both the query response time and the recovery time in case of a fault. A different executable is spawned per city (i.e. Athens and Berlin) so that fault tolerance and resilience can be guaranteed, while each process can run without the complexity of handling different data sets on its memory. The multimodal route planning web service implementation is based on the method of Dibbelt et al. (2012).

The average query service times for the Athens and Berlin instances are reported in Table 1. (time values are averaged over ten executions on a sQEMU Virtual CPU version 1.7.1 clocked at 2.1 GHz, with 4 GB RAM). The execution times surely depend on the instance size and the number of tours to be designed. In all cases the web service derives tour recommendations in less than 1.7 s, which definitely meets the real time application requirements.

5. User evaluation

The tourist tour planner prototype has undergone through official evaluation trials held in September 2014, in Berlin (Germany), in the context of the EU FP7 eCOMPASS project pilot activities. Overall, 47 participants (mostly students and permanent residents of Berlin, hence, largely familiar with the attractions and transit network of the city), have been recruited to participate into the evaluation. The primary purpose of the pilot study has been to evaluate the core functionality of the application, while the main usability & UX evaluation has been conducted by means of an expert review focusing on compliance to usability norms. In particular, the test comprised two phases:

- A heuristic evaluation, based on the DIN EN ISO 9241-11-110,[9] aiming at testing the application's compliance with respect to several dimensions (self-descriptiveness; suitability for the task; controllability; conformity to user expectations; suitability for learning; error tolerance; suitability for individualization).
- A 'scenario walkthrough', focusing on assessing the usability of the application and the meaningfulness of the recommended tours; in that phase, users have been requested to carry out specific tasks; user feedback has been extracted based on a variety of methods (live observation, thinking aloud transcripts and short interviews).

The heuristic evaluation revealed that the prototype application can already be used in an efficient, effective and "satisfying" way. As regards the actual user evaluation, the participants have been asked to plan tours for hypothetical half-day and three-day visits to Berlin, entering their real preferences on POI categories. The look-and-feel, usability and responsiveness (i.e. performance) of the application have been well received by all users, in agreement with the findings of the heuristic evaluation conducted prior to live testing. Furthermore, participants' feedback for the recommended tours has been positive with respect to: quality and attractiveness (POIs included); feasibility (POIs sequencing along itineraries, soundness of multimodal routing directions among POIs); POIs relevance with the users' personal preferences; overall perceived utility of the application in unfamiliar tourist destinations.

Users have also suggested several features to further improve the functionality of the prototype: identification of POIs already visited so that would be excluded in future tour updates/recommendations; designation of POIs that should definitely be incorporated in the recommended tours; bookmarking and/or specifying favorite POIs; map-based visualization of POIs in vicinity to the user's current location. A short (public) version of the eCOMPASS pilot results consolidation report has been published by Marte, Litzenberg, and Krietsch (2014).

6. Conclusion and future research

Expert and intelligent systems inherently require the solution of hard optimization problems to enable intelligent decision making. The problem tackled in this paper is the TTDP, an *NP*-hard combinatorial optimization problem (Gavalas et al., 2014b). Time-dependency (i.e. the consideration of multimodal transfers) represents a realistic condition in the urban tourist tour planning problem, which further increases the complexity of the TTDP due to significantly expanding the problem space. Exact solution approaches, e.g. integer programming, are not appropriate as they would require substantial computational resources and long execution times. For these reasons, alike most expert and intelligent systems, we have proposed a metaheuristic as a solution method. Specifically, a local search heuristic, the SlackRoutes, is employed for solving TTDP, where the main criterion at each insertion step is the selection of the candidate POI which promises to accommodate the most POI insertions along derived tours in future iterations. Unlike the most relevant approach in multimodal tourist tour planning (Garcia et al., 2013), SlackRoutes is not based on the unrealistic assumption of periodic service schedules. SlackRoutes comprises the algorithmic engine of eCOMPASS achieving a fair compromise among deriving high quality solutions (i.e. attractive tours) and responding fairly fast to address the requirements of real-time TTDP applications (Gavalas, 2014). Although more advanced metaheuristics could be used for solving TTDP (such as evolutionary algorithms), we opted for a simpler local search heuristic in order to meet the real-time performance objective. It is also worth mentioning that the SlackRoutes could be applied in a more general setting, e.g., for solving variants of the vehicle routing problem.

To the best of our knowledge, eCOMPASS is the only available research or commercial personalized tour planner that assists the way arounds of tourists through public transit. Further advancing the state of the art of analogous tools, eCOMPASS is the only allowing users to define arbitrary start/end locations for their daily tours, taking into account weather forecast in tour planning and scheduling lunch breaks at appropriate restaurants. The publicly available eCOMPASS web and mobile tools already support tour planning for Athens and Berlin metropolitan areas, showcasing the utility of the system.

In the future we plan to investigate multi-constraint variants of TDTOPTW to capture further constraints dealt with in realistic scenarios, e.g. to allow modeling a maximum budget to be spent while on travel (for both entrance fees and transit fares). We also plan to study an alternative TTDP modeling approach, wherein profits are also assigned to arcs (in addition to nodes) so as to incorporate routes of scenic value within recommended tours; the mixed Orienteering Problem (Gavalas, Konstantopoulos, Mastakas, Pantziou, & Vathis, 2015) could serve as a baseline problem to address this requirement. Last, we will investigate machine learning approaches to account for uncertainty dealt with in transit services schedules; this will enable robust tour planning, which represents a critical consideration, especially when considering late hours and/or infrequent services.

Acknowledgments

This work has been supported by the EU FP7/2007–2013 Programme under Grant agreements No. 288094 (eCOMPASS) and No. 621133 (HoPE). The authors would also like to acknowledge the support of J. Dibbelt, A. Bauer and T. Pajor from the Karlsruhe Institute of Technology (KIT) who kindly provided their multimodal route planning web service implementation.

References

Bolzoni, P., Helmer, S., Wellenzohn, K., Gamper, J., & Andritsos, P. (2014). Efficient itinerary planning with category constraints. In *Proceedings of the 22nd ACM*

[9] The ISO 9241-11-110 focuses on the ergonomics and usability of human–system interaction.

SIGSPATIAL international conference on advances in geographic information systems (pp. 203–212).
Borràs, J., Moreno, A., & Valls, A. (2014). Intelligent tourism recommender systems: A survey. *Expert Systems with Applications, 41*(16), 7370–7389.
Brilhante, I. R., Macedo, J. A., Nardini, F. M., Perego, R., & Renso, C. (2015). On planning sightseeing tours with TripBuilder. *Information Processing & Management, 51*(2), 1–15.
Brown, B., & Chalmers, M. (2003). Tourism and mobile technology. In *Proceedings of the eighth European conference on computer supported cooperative work* (pp. 335–354).
Cohen, R., & Katzir, L. (2008). The generalized maximum coverage problem. *Information Processing Letters, 108*(1), 15–22.
De Choudhury, M., Feldman, M., Amer-Yahia, S., Golbandi, N., Lempel, R., & Yu, C. (2010). Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM conference on hypertext and hypermedia* (pp. 35–44).
Dibbelt, J., Pajor, T., & Wagner, D. (2012). User-constrained multi-modal route planning. In *Proceedings of the 14th meeting on algorithm engineering and experiments* (pp. 118–129).
Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8*(2), 67–71.
Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., & Linaza, M. T. (2013). Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research, 40*(3), 758–774.
Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., & Vathis, N. (2014). Efficient heuristics for the time dependent team orienteering problem with time windows. In *Proceedings of the international conference on applied algorithms* (pp. 151–162).
Gavalas, D., Konstantopoulos, C., Mastakas, K., & Pantziou, G. (2014a). Mobile recommender systems in tourism. *Journal of Network and Computer Applications, 39*, 319–333.
Gavalas, D., Konstantopoulos, C., Mastakas, K., & Pantziou, G. (2014b). A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics, 20*(3), 291–328.
Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., & Vathis, N. (2015). Approximation algorithms for the arc orienteering problem. *Information Processing Letters, 115*(2), 313–315.
Gionis, A., Lappas, T., Pelechrinis, K., & Terzi, E. (2014). Customized tour recommendations in urban areas. In *Proceedings of the seventh ACM international conference on web search and data mining* (pp. 313–322).
Han, Y., Guan, H., & Duan, J. (2014). Tour route multiobjective optimization design based on the tourist satisfaction. *Discrete Dynamics in Nature and Society*. 2014.
Hasuike, T., Katagiri, H., Tsubaki, H., & Tsuda, H. (2014). Route planning problem with groups of sightseeing sites classified by tourist's sensitivity under time-expanded network. In *Proceedings of the 2014 IEEE international conference on systems, man and cybernetics* (pp. 188–193).
Hu, Q., & Lim, A. (2014). An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research, 232*(2), 276–286.
Kurata, Y., & Hara, T. (2014). CT-Planner4: Toward a more user-friendly interactive day-tour planner. In *Proceedings of the 2014 international conference on information and communication technologies in tourism* (pp. 73–86).
Labadi, N., Melechovský, J., & Wolfler Calvo, R. (2011). Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics, 17*, 729–753.
Laporte, G., & Martello, S. (1990). The selective travelling salesman problem. *Discrete Applied Mathematics, 26*(2), 193–207.
Lew, A., & McKercher, B. (2006). Modeling tourist movements: A local destination analysis. *Annals of Tourism Research, 33*(2), 403–423.
Likas, A., Vlassis, N., & Verbeek, J. (2003). The global *k*-means clustering algorithm. *Pattern Recognition, 36*(2), 451–461.
Lim, K. H. (in press). Recommending tours and places-of-interest based on user interests from geo-tagged photos. In *Proceedings of SIGMOD'2015*.
Marte, M., Litzenberg, B., & Krietsch, F. (2014). Pilot results consolidation. *Deliverable D6.2.2, EU FP7 eCOMPASS*, <http://www.ecompass-project.eu/sites/default/files/eCOMPASS-Deliverable-D6.2.2-v1.3_0.pdf>.
Schaller, R., & Elsweiler, D. (2014). Itinerary recommenders: How do users customize their routes and what can we learn from them? In *Proceedings of the fifth information interaction in context symposium* (pp. 185–194).
Souffriau, W., & Vansteenwegen, P. (2010). Tourist trip planning functionalities: State of the art and future. In *Proceedings of the 10th international conference on current trends in web engineering* (pp. 474–485).
Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research, 209*(1), 1–10.
Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., & Van Oudheusden, D. V. (2009). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research, 36*, 3281–3290.
Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., & Van Oudheusden, D. (2011). The city trip planner: An expert system for tourists. *Expert Systems with Applications, 38*(6), 6540–6546.