A Personalized Multimodal Tourist Tour Planner

Damianos Gavalas
Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Greece
&
CTI, Patras, Greece
dgavalas@aegean.gr

Vlasios Kasapakis
Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Greece
&
CTI, Patras, Greece
v.kasapakis@aegean.gr

Charalampos Konstantopoulos
Department of Informatics, University of Piraeus, Piraeus, Greece
&
CTI, Patras, Greece
konstant@unipi.gr

Grammati Pantziou
Department of Informatics, Technological Educational Institution of Athens, Athens, Greece
&
CTI, Patras, Greece
pantziou@teiath.gr

Nikolaos Vathis
School of Electrical and Computer Engineering, NTUA, Athens, Greece
&
CTI, Patras, Greece
nvathis@softlab.ntua.gr

Christos Zaroliagis
Department of Computer Engineering & Informatics, University of Patras, Patras, Greece
&
CTI, Patras, Greece
zaro@ceid.upatras.gr

ABSTRACT
Tourists become increasingly dependent on mobile city guides to locate tourist services and retrieve information about nearby points of interest (POIs) when visiting unknown destinations. Although several city guides support the provision of personalized tour recommendations to assist tourists visiting the most interesting attractions, existing tour planners only consider walking tours. Herein, we introduce eCOMPASS, a context-aware mobile application which also considers the option of using public transit for moving around. 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 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. This paper describes the routing algorithm which comprises the core functionality of eCOMPASS and discusses the implementation details of the mobile application using the metropolitan area of Berlin (Germany) as case study.

Categories and Subject Descriptors
G.2.1 [**DISCRETE MATHEMATICS**]: Combinatorics – *Combinatorial algorithms, Permutations and combinations.* H.4.m [INFORMATION SYSTEMS APPLICATIONS]: Miscellaneous

General Terms
Algorithms, Design, Verification.

Keywords
Tourist Trip Design Problem, Multimodal Route Planning, Orienteering Problem, Context Awareness, Mobile Application.

This work has been supported by the EU FP7/2007-2013 Programme under grant agreements no. 288094 (eCOMPASS), no. 609026 (MOVESMART) and no. 621133 (HoPE).

MUM '14, November 25 - 28 2014, Melbourne, VIC, Australia
Copyright 2014 ACM 978-1-4503-3304-7/14/11…$15.00
http://dx.doi.org/10.1145/2677972.2677977

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 [1]. 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 the public transit systems, thereby making transit transfers a cumbersome exercise [7].

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) [4]. 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.

Notably, most algorithmic approaches addressing TTDP assume constant travel times among POIs, i.e. they consider exclusively walking transfers. Such approaches overlook the real aspects of tourist movement patterns which entail the use of public transportation to cover overly long distances within tourist areas [7].

Herein, we propose a novel cluster-based heuristic approach, the *SlackRoutes*, which addresses the above described shortcomings of existing approaches to TTDP. 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 also 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 [5].

SlackRoutes comprises the core functionality of *eCOMPASS*, a context-aware mobile application deriving personalized daily tourist itineraries. eCOMPASS advances the state of the art (among all known research prototypes and commercial tools) with respect to several aspects:

- While existing tourist itinerary planners only consider walking tours, eCOMPASS takes into account time-dependent travel times among POIs, namely it considers the option of using public transit for moving around. 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 time available for sightseeing and minimize waiting time at transit stops.
- To ensure fast response to user queries, existing tools restrict users to select the start/end points of their daily itineraries among a fixed set of locations (typically a list of accommodations and/or landmarks). eCOMPASS follows a novel approach that overcomes this restriction and allows users to define arbitrary start/end locations (e.g. mobile users may choose to start their itinerary from their current location and end it anywhere else).

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 concludes our work and suggests directions for future research.

2. RELATED WORK

2.1 Algorithmic approaches to TTDP

TTDP has received considerable attention in the recent years with several -mainly heuristic- algorithmic methods proposed to solve it [4]. 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 24x7 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. The baseline combinatorial optimization problem for TTDP is the Orienteering Problem (OP) [11]. 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.

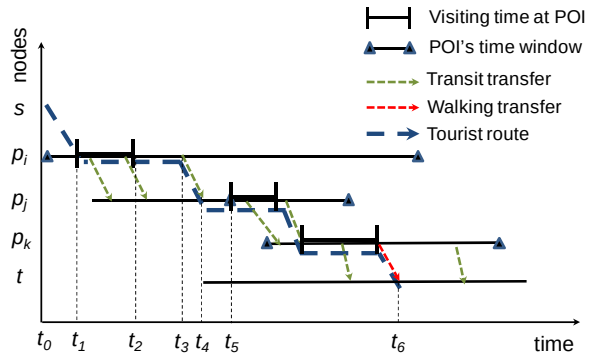

Figure 1: Illustration of a tourist path (blue dashed line) from s to t via POIs p_i, p_j and p_k. Green dashed arrows indicate available multimodal transfer options among POIs.

Clearly, the OP may be used to model the simplest version of the TTDP wherein the POIs are associated with a profit (i.e. user satisfaction) and the goal is to find a single tour that maximizes the profit collected within a given time budget (time allowed for sightseeing in a single day). Extensions of the OP have been successfully applied to model more complex versions of the single tour TTDP. The team orienteering problem (TOP) represents the extension of the OP to multiple tours. The TOP with time windows (TOPTW) considers visits to locations within a predefined time window (this allows modeling opening and closing hours of POIs). The time-dependent TOPTW (TDTOPTW) considers time dependency in the estimation of time required to move from one location to another and, therefore, it is suitable for modeling multi-modal transports among POIs. Figure 1 illustrates the example output of a TDTOPTW solver (for a single tourist path).

TOPTW has been mostly commonly studied among the aforementioned OP variants [4], since it is useful for modeling several real-life optimization problems. Among existing TOPTW

solvers, the iterated local search (ILS) algorithm [10] represents a fair compromise in terms of speed (less than 7 sec for up to 200 POIs and k=4 daily tours) versus deriving routes of reasonable quality (on average, less than 5% gap from the best known solution). As a result, ILS is considered most suitable for real-time TTDP applications among alternative TOPTW algorithms. On the downside, ILS treats each POI separately, thereby commonly overlooking highly profitable areas of POIs situated far from current location considering them too time-expensive to visit. ILS is also often trapped in areas with isolated high-profit POIs, possibly leaving considerable amount of the overall time budget unused [5].

The work of Garcia et al. [3] is the first to address the TDTOPTW algorithmically. The authors presented two different approaches to solve TD-TOPTW, both applied on real urban test instances (POIs and bus network of San Sebastian). 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 insertion phase part of ILS. In case that the derived TOPTW solution is infeasible (due to violating the time windows of nodes included in the solution), a number of visits are removed. The second approach uses time-dependent travel times but it based on the simplified assumption of periodic service schedules; this assumption does not hold in realistic urban transportation networks. Herein, we propose SlackRoutes, an algorithmic approach that relaxes this assumption and is applicable to realistic transit networks.

2.2 Web and mobile TTDP solvers

Among the many available mobile tourist guides, three web/mobile tools are the only ones known to offer tour planning services: CT-Planner4[1], CityTripPlanner[2] and mtrip[3]. These tools, essentially TTDP solvers, automate the creation of a single or multiple tours via a set of POIs taking into account their respective profit, visiting time and opening hours as well as the walking travel times among POIs and the trip details (visiting days, start/end times). The derived tours are personalized, i.e. they are tuned according to user-defined preferences.

CT-Planner4 [6] is a web-based tourist tour planner for seven (7) Japanese cities. Recommended tours are personalized with respect to: (a) user focus and taste, which adjust POI profits, and (b) preferred moving speed and reluctance to walk, which adjust walking travel times. The tour planning engine of CT-Planner4 relies on a genetic algorithm which solves the Selective Traveling Salesman Problem (STSP), i.e. it derives a single tourist itinerary. The algorithm's parameters are tuned so that the computation completes within a second.

CityTripPlanner [12] is a web/mobile city tour planner which already covers 76 destinations in all continents. An Android/iOS mobile application may be downloaded for free, which contains the core functionality; the actual city tour is generated, downloaded and purchased from the website. Recommended tours are displayed on a map and list view. The user is allowed to edit derived tours, remove unwanted POIs and/or adjusting 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. CityTripPlanner is known to utilize the ILS TOPTW algorithm [10] as its core tour planning engine.

mtrip is a mobile app that currently covers 28 destinations in Europe, Asia and America. It is the only mobile tour planner offering augmented reality views of POIs (namely, it superimposes POI markers upon the smartphone's camera views). mtrip is also the only tour planning tool known to function offline, namely map data are incorporated within the purchased, standalone Android/iOS mobile application. Being a commercial prototype, no technical details are provided regarding the algorithmic machinery used to derive personalized tours for mtrip.

The aforementioned city tour planning software tools already incorporate an array of useful services. However, they 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, the reviewed tools allow users to select the start/end points of their itineraries among a fixed set of lodging and/or landmark locations. Rather, practical scenarios would likely involve users requesting tours starting/ending at arbitrary locations (the starting point would most probably be their location at query time).

Our proposed tourist tour planner addresses all the above mentioned challenges. In particular, the support of tourists moving around either walking or using public transit represents a focal design objective of eCOMPASS tour planning engine. 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, our proposed tool allows users to define -different- arbitrary start/end locations for their daily tours.

3. THE SLACKROUTES TOUR PLANNING ALGORITHM

SlackRoutes is a TDTOPTW solver which comprises the algorithmic core of eCOMPASS. SlackRoutes is given a complete directed graph $G = (V, E)$ where V denotes the set of locations with $N = |V|$; a set $P = \{p_1, p_2, \dots p_{|P|}\} \subseteq V$, denoting the set of POIs. The algorithm 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 r^{th} route are denoted as s_r and t_r, respectively. Accordingly, st_r and et_r denote the starting, ending time, respectively of the r^{th} route, $r = 1, 2, \dots m$.

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 $\text{TW}_{ir} = [\text{open}_{ir}, \text{close}_{ir}]$; $r = 1, 2, \dots m$ (a POI may have different time windows per day).

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: $tr_{u,v}(t) = \min\{walking_{u,v}, delay_{u,v}(t) + travtime_{u,v}(t)\}$. The attribute $walking_{u,v}$ denotes the time required to walk 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; $travtime_{u,v}(t)$ is the total travel time spent after the initial

[1] http://ctplanner.jp/ctp4/index-e.html

[2] http://www.citytripplanner.com/

[3] http://www.mtrip.com/

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 S_{uv}, which contains all the non- pairs ($dep^i_{u,v}$, $tr^i_{u,v}$); i = 1, 2, … $|S_{uv}|$, in ascending dominated order of $dep_{u,v}(t)$, where $dep_{u,v}(t)$ is a departure time and $tr_{u,v}(t)$is the corresponding travel time from u to v. We consider that a departure travel time pair from node u to node v (dep, tr) dominates a pair (dep', tr') iff $dep > dep'$ $dep' + tr' \leq dep + tr$.

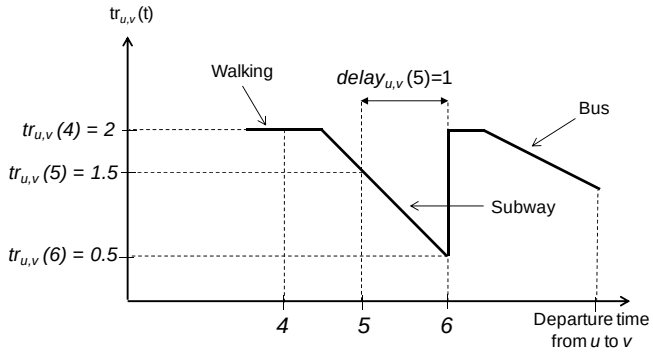

Figure 2. Traveling time from u to v as a function of the departure time from u (travel time profile of $u \rightarrow v$).

Figure 2 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 travelling 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 time the visit at p_i completes, i.e., the departure time from p_i; $leave_i = start_i + visit_i$.

$arrive_i$ The arrival time at p_i; $arrive_i = leave_{prev(i)} + tr_{prev(i),i}(leave_{prev(i)})$ where $leave_{prev(i)}$is the departure time from the previous node of p_i in route r (prev(i)). We assume that $\text{arrive}_{s_r} = st_r$.

$maxStart_i$ The latest time the visit at p_i can start without violating the time windows of the nodes following p_i; $maxStart_i = min\{close_{ir}, \max\{t: t + tr_{i,next(i)}(t) \leq maxStart_{next(i)}\} - visit_i\}$, where next($i$) is the node following p_i in r. We assume that $\text{maxStart}_{t_r} = et_r$.

$slack_i$ How long the arrival at p_i may be delayed without affecting the feasibility of following visits, i.e. the time available for inserting a new POI before p_i; $slack_i = maxStart_i - arrive_i$. Note that if the value of $slack_i$ is close to 0, it is highly unlikely the insertion of a new POI between $p_{\text{prev}(i)}$ and p_i to be feasible

It is noted that the variables *wait*, *start* and *arrive* have been defined in the context of ILS algorithm [10]. 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
4.      RouteInit (ClusterSet)
5       iterations ← 0
6.      while iterations < maxIterations do
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
```

Preprocessing phase

The preprocessing phase contributes to the efficient handling of user queries, thereby meeting real-time application requirements. 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 Figure 3a, 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 (see Figure 3b), using the global k-means algorithm [8].

The main incentives behind the clustering process are: (a) to utilize the clustered topology in the online phase so as to mainly consider successive visits to POIs grouped in the same cluster; 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 topology areas featuring high density of 'promising' (i.e. highly profitable) POIs.

Online execution phase

The online execution phase is triggered upon the receipt of user queries which convey user 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 (for instance, in Figure 3c, 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 shakes derived solutions in hope of escaping local optima and achieving further improvement (line 13). Among all solutions, the one 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, used to measure impact of the published work of scholars; for instance a cluster with $C_q = 10$ should include at least 10 POIs with profit greater or equal to 10. The intuition behind the C_q ordering is to motivate visits to clusters highly dense in POIs with high profit values (i.e. clusters with large C_q), even if those POIs are located relatively far. A number of *m* clusters are popped from listOfClusters, starting with those with the highest C_q value. Then, one POI is inserted into each of the originally empty *m* routes with each of those POIs coming from a different cluster (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 the route feasibility criterion is satisfied (i.e. 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 at each feasible position *k* along *r*, that is after any POI p_{i_k}[4]. For each feasible position, it calculates the weight w_i^k of p_i:

$$w_i^k = profit_i \cdot A_i^k, \quad (1)$$

$$A_i^k = \frac{\sum_{j=1}^{i} slack_j + slack_k + \sum_{j=i+1}^{n'} slack_j}{n'+1} \quad (2)$$

The quantity A_i^k denotes the average slack value among all POIs on *r*, after the insertion of p_i after the k^{th} 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. It is also worth mentioning 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 trip (that is, prior and after visiting p_i). Eventually, the POI with the highest insertion weight w_i, where $w_i = \max(w_i^k)$, is inserted into the position *k* which maximizes its weight. Notably, unlike most existing iterated local search procedures, e.g. the ILS algorithm [10], 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.

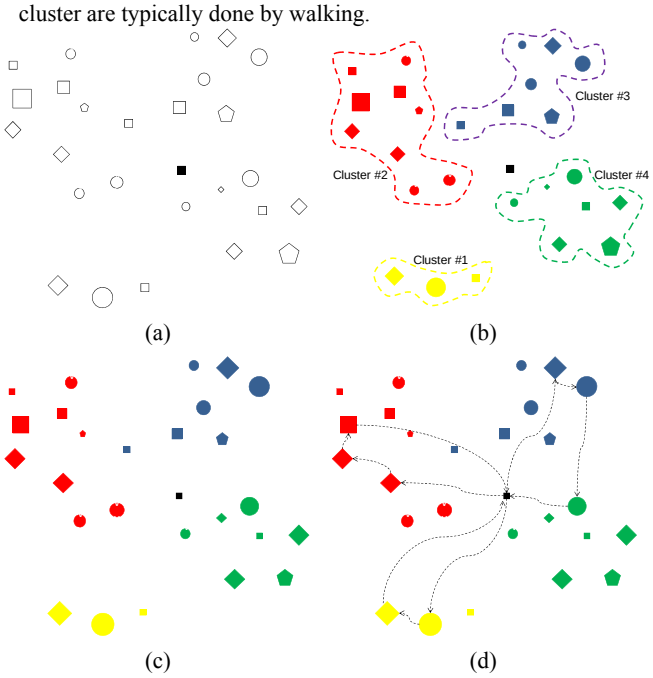

Figure 3: Illustration of SlackRoutes execution phases: (a) initial topology (geometric shapes indicate POI categories, while their size denote POIs profit values); (b) clustering of POIs based on geographic criteria (preprocessing phase); (c) adjustment of POI profit values based on user profile (i.e. user preferences upon POI categories indicated through user queries); (d) example output (tours) derived by SlackRoutes.

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 Figure 3d.

3.2 Support for arbitrary start/end itinerary locations

(TD)TOPTW modeling involves an asymmetric distance matrix which stores travel times among all nodes. The distance matrix should be precomputed to ensure fast response (tour creation) to user queries. The offline calculation becomes more crucial when considering time dependent travel times (i.e. full travel time profiles similar to that shown in Figure 2), 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*).

[4] If *k=0*, the POI *i* is placed first of all POIs of the route. A POI may be inserted at any location that ensures that any cluster is visited only once within any particular route. Hence, a POI can be inserted prior or after a POI of its own cluster, if such a POI exists, otherwise it can only be inserted between POIs of separate clusters. This restriction saves execution time by limiting the solution space and reduces transit transfers, since intra-cluster transfers are typically done by walking.

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.

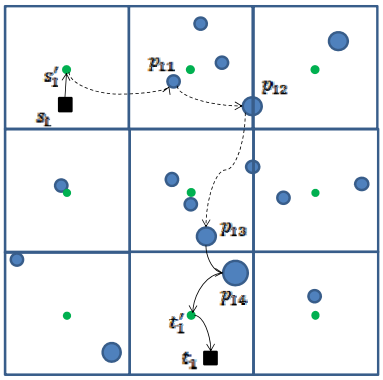

Figure 4. Consideration of arbitrary start/end locations; solid and lines denote walking and dashed lines transit transfers.

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. 500m x 500m), covering the whole geographical area where POIs are located in. Within each region R_i a central location is chosen, called region center c_{R_i}. For instance, in Figure 4, 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 V, the travel profile is calculated.

Then, 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 fairly fast to compute the walking distances, while walking is highly likely to be 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).
- Execute the SlackRoutes algorithm and connect (s_i, s'_i) and (t_i, t'_i) as first/last route legs.
- 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.
- Repair the route inserting new POIs, if possible.

4. eCOMPASS system implementation

This Section focuses on the implementation details of eCOMPASS. Section 4.1 describes the structure of the tourist content database. Section 4.2 explains how multimodal travel profiles are calculated. Section 4.3 presents the system architecture and describes the workflow within the eCOMPASS framework. Section 4.4 discusses implementation details for the web services executed within the eCOMPASS system.

4.1 Content database

We have compiled a collection from the urban area of Berlin (Germany), which comprises 144 (=12x12 km^2) region centers, 113 attractions (POIs) and 100 hotels. POIs are classified in the following categories: museums & art galleries, nature, archaeological sites, neighborhoods & squares, churches & religious heritage, monuments & landmarks. The metadata stored for each POI include: title; geo-coordinates (latitude, longitude); category; profit; visiting time; opening/closing hours for each week day; indication whether it is 'open air' (i.e. unsuitable to visit in rainy or heat wave days) or not; entrance fee for adults and children; indication for accessibility facilities; short description; photograph(s); address; telephone; official website URL; Wikipedia entry URL; average user rating, overall number of uploaded user ratings.

Profits have been set in a 1-100 scale and visiting times vary from 5 minute (e.g. for some outdoor statues) to 2 hours (e.g. for some not-miss museums and wide-area archaeological sites). About half of the POIs are outdoors and always visitable (24h time windows) while the remainder are associated with relatively wide, largely overlapped time windows (typically around 8h). Most of selected hotels are situated around main attraction areas, while some hotels are situated in city suburbs.

4.2 Calculation of multimodal travel time profiles

In our implementation, we have used the GTFS[5] (General Transit Feed Specification) transit network data of the Berlin metropolitan area; the transit network includes 3213 stops, 9 lines U-Bahn, 15 lines S-Bahn, 22 tram lines and 147 bus lines. Using the method of Dibbelt et al. [2], we compute offline pairwise full (24h range) multimodal time-dependent travel time profiles among all locations stored in the content database (POIs, region centers, hotels, restaurants, cafes). The overall shortest time dependent travel time information is pre-calculated and stored in a three-dimensional array of size N x N x 1440, where N is the number of specified locations/POIs and 1440 (=24x60) 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 5.76 GB.

4.3 eCOMPASS architecture and application workflow

eCOMPASS adopts a service-oriented architecture (SOA) approach, wherein the business logic is composed of loosely coupled web services, combined together to deliver personalized services either through traditional web interfaces or thin mobile clients. The mobile client application (https://www.youtube.com/watch?v=BVT__mWOoso) has been developed on the Android platform and is available from http://ecompass.aegean.gr/eCompass.apk.

Figure 5 provides a high-level overview of the eCOMPASS architecture as well as the procedural steps taking place 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 4.2). Also, POIs are

[5] https://developers.google.com/transit/gtfs/reference

grouped in disjoint clusters based on geographical criteria.

2. POIs metadata (see Section 4.1) and travel profiles among POIs (see Section 4.2) are stored in memory structures on the server side.

Online phase:

3. User queries are sent along with the user profile (preferences among individual POI categories) and trip detail.
4. A weather web service (Yahoo! weather[6]) 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).
5. 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 3.2).
6. The tour planning web service derives the personalized daily tourist tours.
7. 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.

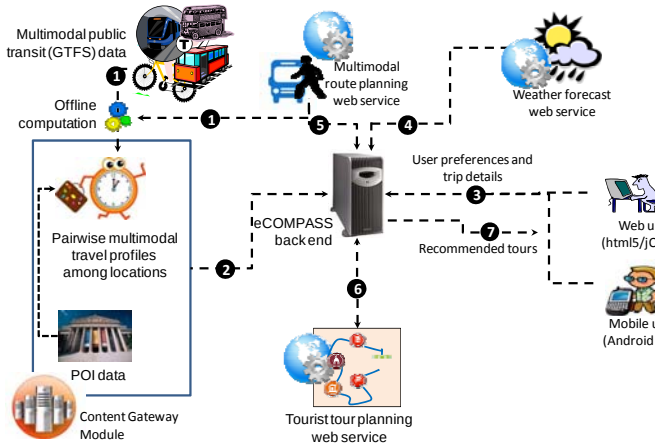

Figure 5. eCOMPASS system overview.

Figure 6 exhibits representative screens captured by the eCOMPASS mobile client application. The main application features are highlighted in the menu of Figure 6a; those include listings of attractions, hotels, restaurant and cafes further to the main tour planning functionality.

The next screen presents the user settings used to personalize the derived tourist tours (see Figure 6b). The user specifies the start and end location of each itinerary, allowed to choose among available hotels (since most itineraries are expected to start and end at the user's accommodation), 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 and the preferred walking pace (to adjust the estimated walking travel times).

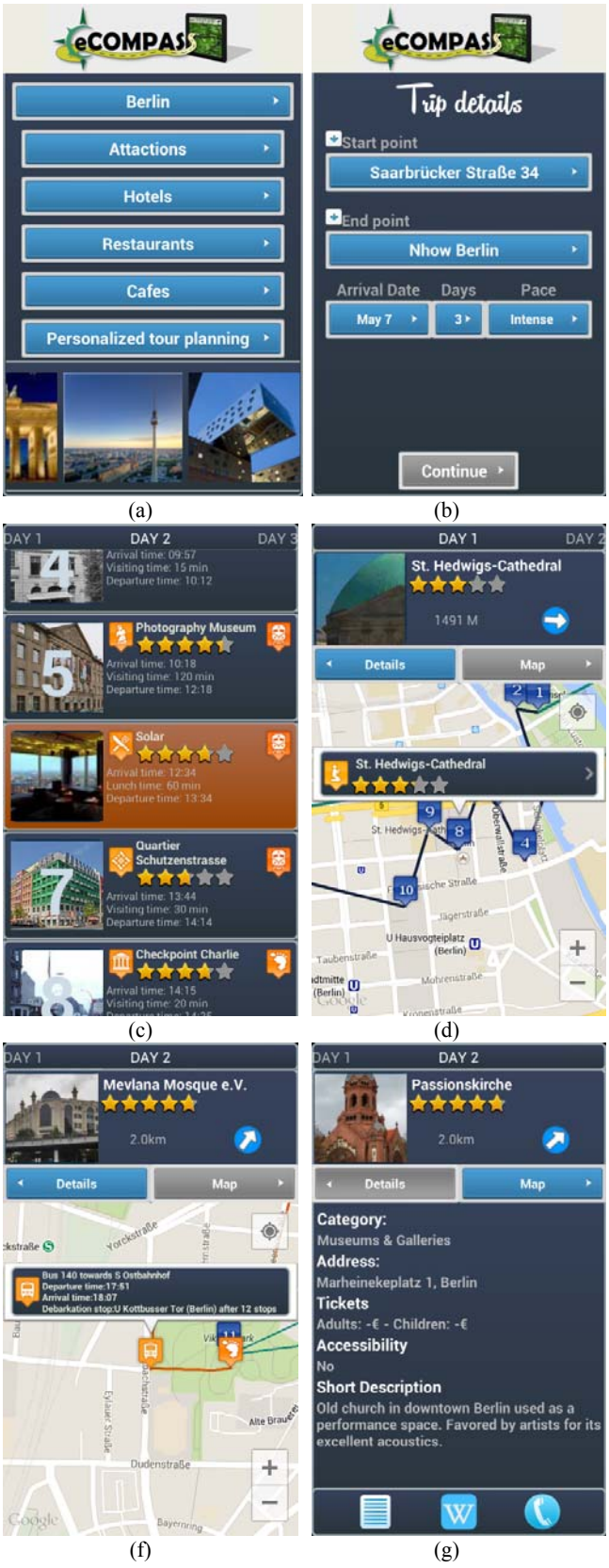

Figure 6. Screenshots taken from the eCOMPASS mobile (Android) client application.

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

Recommended tours may be visualized in both list (see Figure 6c) and map (see Figure 6d) 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 Figure 6e). The user may retrieve further information for selected POIs, including address, telephone, entrance fee, accessibility facilities, short description (see Figure 6f). 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 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 rest of them. On the incident of modifying any user setting, the tour planning web service is reinvoked and the itineraries are recalculated.

4.4 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. The core tourist tour planning service is invoked through a HTTP GET request formatted as follows:

```
/tours/json/?[{date}{start_lon}{start_lat}{end_lon}
    {end_lat}{start_time}{end_time}{[excluded_POIs]}
    {[userprefs]}]
```

In particular, for each daily tour the tour planning service receives the following parameters:

- start_lon: longitude of the tour start location;
- start_lat: latitude of the tour start location;
- end_lon: longitude of the tour end location;
- end_lat: latitude of the tour end location;
- start_time: the tour start time (in min)
- end_time: the tour end time (in min);
- excluded_POIs: array of POIs not to be considered (either explicitly excluded by the user or due to unsuitable weather conditions);
- userprefs=array of ratings for each POI category (in 1-10 scale).

The average query service times for Berlin instances vary from 400 msec (1 tour) to 1,65 sec (4 tours); 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.

5. CONCLUSIONS

In this paper, we introduced eCOMPASS, a context-aware mobile application which derives personalized multimodal tours via selected urban attractions. To the best of our knowledge, eCOMPASS is the only available research or commercial tour planner that assists the way arounds of tourists through public transit. The algorithmic core of eCOMPASS is SlackRoutes, an efficient TTDP solver, which takes into account time dependency in estimating travel times between urban locations and derives near-optimal tours so as to best utilize time available for sightseeing and minimize the time spent at transit stops. Compared to analogous tools, eCOMPASS is the only allowing users to define arbitrary start/end locations for their daily tours and taking into account weather forecast in tour planning. The publicly available eCOMPASS mobile tool supports tour planning for Berlin metropolitan area, showcasing the utility of the system.

6. ACKNOWLEDGMENT

The authors acknowledge the support of J. Dibbelt, A. Bauer and T. Pajor from Karlsruhe Institute of Technology (KIT) who kindly provided their multimodal route planning web service implementation.

7. REFERENCES

[1] Brown B., & Chalmers, M. 2003. Tourism and mobile technology. *Proceedings of the 8th European Conference on Computer Supported Cooperative Work* (ECSCW'03), 335-354.

[2] Dibbelt, J., Pajor, T. and Wagner, D. 2012. User-constrained multi-modal route planning. *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments* (ALENEX'12), 118-129.

[3] Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W. and Linaza, M. T. 2013. Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research*, 40(3), 758-774.

[4] Gavalas, D., Konstantopoulos, C., Mastakas, K. and Pantziou, G. 2014. A Survey on Algorithmic Approaches for Solving Tourist Trip Design Problems. *Journal of Heuristics*, 20(3), 291-328.

[5] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N. 2014. Efficient Heuristics for the Time Dependent Team Orienteering Problem with Time Windows. *Proceedings of the International Conference on Applied Algorithms* (ICAA'2014), 151-162.

[6] Kurata, Y. and Hara, T. 2014. CT-Planner4: Toward a More User-Friendly Interactive Day-Tour Planner. *Proceedings of the International Conference on Information and Communication Technologies in Tourism 2014* (ENTER'2014), 73-86.

[7] Lew, A. and McKercher, B. 2006. Modeling tourist movements: A local destination analysis. *Annals of Tourism Research*, 33(2), 403-423.

[8] Likas, A., Vlassis, N., & Verbeek, J. 2003. The global k-means clustering algorithm. *Pattern Recognition*, 36(2), 451-461.

[9] McKean, J., Johnson, D. and Walsh, R. 1995. Valuing time in travel cost demand analysis: An empirical investigation. *Land Economics*, 71: 96-105.

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

[11] Vansteenwegen, P., Souffriau, W. and Van Oudheusden, D. 2011. The orienteering problem: a survey. *European Journal of Operational Research*, 209(1), 1-10.

[12] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G. and Van Oudheusden, D. 2011. The city trip planner: an expert system for tourists. *Expert Systems with Applications*, 38(6), 6540-6546.