Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 026

Efficient Heuristics for the Time Dependent Team Orienteering Problem with Time Windows

D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou and N. Vathis

January 2014

Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 026

Efficient Heuristics for the Time Dependent Team Orienteering Problem with Time Windows

D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou and N. Vathis

January 2014

Efficient Heuristics for the Time Dependent Team Orienteering Problem with Time Windows*

Damianos Gavalas[1,6], Charalampos Konstantopoulos[2,6], Konstantinos Mastakas[3,6], Grammati Pantziou[4,6], and Nikolaos Vathis[5,6]

[1] Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Greece, dgavalas@aegean.gr
[2] Department of Informatics, University of Piraeus, Piraeus, Greece, konstant@unipi.gr
[3] Department of Mathematics, University of Athens, Athens, Greece, kmast@math.uoa.gr
[4] Department of Informatics, Technological Educational Institution of Athens, Athens, Greece, pantziou@teiath.gr
[5] School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece, nvathis@softlab.ntua.gr
[6] Computer Technology Institute and Press "Diophantus" (CTI), Patras, Greece

Abstract. The Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW) can be used to model several real life problems. Among them, the route planning problem for tourists interested in visiting multiple points of interest (POIs) using public transport. The main objective of this problem is to select POIs that match tourist preferences, while taking into account a multitude of parameters and constraints and respecting the time available for sightseeing in a daily basis. TDTOPTW is NP-hard while almost the whole body of the related literature addresses the non time dependent version of the problem. The only TDTOPTW heuristic proposed so far is based on the assumption of periodic service schedules. Herein, we propose two efficient cluster-based heuristics for the TDTOPTW which yield high quality solutions, take into account time dependency in calculating travel times between POIs and make no assumption on periodic service schedules. The validation scenario for our prototyped algorithms included the metropolitan transit network and real POI sets compiled from Athens (Greece).

1 Introduction

The aim of the Team Orienteering Problem with Time Windows (TOPTW) is to maximize the total profit collected by visiting a set of locations, each of which has a profit, a service time and a time window. The number of routes is limited, and each location can be visited at most once. The TOPTW has numerous real-life

* This work was supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS).

applications. In this paper, we consider the route planning application for tourists interested in visiting multiple POIs. The main objective is to select POIs that match tourist preferences, while taking into account a multitude of parameters and constraints (distances among POIs, visiting time required for each POI, POIs' opening hours) and respecting the time available for sightseeing in a daily basis. The problem is further complicated when considering the complexity of metropolitan transit networks commonly used by tourists to move from a POI to another. In this case, the tourist route planning problem can be modeled as a TDTOPTW. To our knowledge, the first TDTOPTW heuristic has been recently proposed by Garcia et al. [1]. However, the proposed algorithm is based on the simplified assumption of periodic service schedules which clearly, is not valid in realistic transportation networks, wherein arrival/departure frequencies typically vary within the services operational periods.

Herein, we propose two novel heuristic approaches, the Time Dependent CSCRoutes (TDCSCRoutes) and the SlackCSCRoutes, which address the above described shortcoming of the existing approach to TDTOPTW. The main incentive behind our approaches is to motivate visits to topology areas featuring high density of highly profitable candidate POIs, while taking into account time dependency (i.e. multimodality) in calculating travel times from one POI to another; the aim is to derive high quality routes (i.e. maximizing the total collected profit) while not sacrificing the time efficiency required for online applications. Our prototyped algorithms have been tested in terms of various performance parameters (solutions quality, execution time, number of transit transfers, etc) upon real test instances compiled from the wider area of Athens, Greece; the calculation of time dependent travel times has been carried out over the Athens metropolitan transit network. The performance of our algorithms has been compared against two variants that use precalculated average travel times (among the individual time dependent, real travel times) between POIs.

The remainder of this article is organized as follows: Section 2 overviews the related work while Section 3 presents our novel cluster-based heuristics. Section 4 discusses the experimental results and finally Section 5 concludes our work.

2 Related work

The TOPTW is an extension of the orienteering problem (OP) Vansteenwegen et al. [2]. In the OP, several locations with an associated score have to be visited within a given time limit. Each location may be visited only once, while the aim is to maximize the overall score collected on a single tour. The team orienteering problem (TOP) extends the OP considering multiple routes while the TOP with time windows (TOPTW) considers visits to locations within a predefined time window. TOPTW is NP-hard (e.g. see [3]). Hence, exact solutions for TOPTW are feasible for instances with very restricted number of locations. As a result, the main body of TOPTW literature exclusively involves heuristic algorithms ([4], [5], [6], [7], [8], [9], [10]). ACS [8], Enhanced ACS [4] and the approach of Tricoire et al. [9] are known to yield the highest quality solutions. The most

efficient known heuristic is based on Iterated Local Search (ILS) [10], offering a fair compromise with respect to execution time versus deriving routes of reasonable quality [2]. However, 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. In [11] two cluster-based extensions to ILS have been proposed to address the aforementioned weakness by grouping POIs on disjoint clusters, thereby making visits to such POIs more attractive.

Time Dependent OP (TDOP) was introduced by Formin and Lingas [12]. TDOP is MAX-SNP-hard since a special case of TDOP, time-depenent maximum scheduling problem is MAX-SNP-hard [13]. Fomin and Lingas [12] give a $(2+\epsilon)$ approximation algorithm for rooted and unrooted TDOP. Abbaspour et al. [14] investigated a variant of Time Dependent OP with Time Windows (TDOPTW) in urban areas, and proposed a genetic algorithm for solving the problem. The work of Garcia et al. [1] is the first to address algorithmically the TDTOPTW. The authors presented two different approaches to solve TDTOPTW, both applied on real urban test instances (POIs and bus network of San Sebastian, Spain). 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, clearly, does not hold in realistic urban transportation networks, especially on non fixed-rail services (e.g. buses). Herein, we propose an algorithmic approach that relaxes this assumption and is applicable to realistic transit networks.

3 The proposed TDTOPTW heuristics

TDTOPTW is an extension of TOPTW integrating public transportation, i.e., time dependent travel costs among nodes. In TOPTW we are given a complete directed graph $G = (V, E)$ where V denotes the set of locations with $N = |V|$; a set $P = \{p_1, p_2, \ldots, p_{N_p}\} \subseteq V$ denoting the set of POIs; an integer m denoting the number of days the trip shall last, and a time budget B. The main attributes of each node $p_i \in P$ are: the service or visiting time (visit_i), the profit gained by visiting p_i (profit_i), and each day's time window $[\text{open}_{ir}, \text{close}_{ir}], r = 1, 2, \ldots, m$, (a POI may have different time windows per day). Every link $(u, v) \in E$ denotes the transportation link from u to v and is assigned a travel time. The objective is to find m disjoint routes each starting from a starting location $s \in V$ and ending at a location $t \in V$, each with overall duration limited by the time budget B, that maximize the overall profit collected by visited POIs in all routes. TDTOPTW is an extension of TOPTW where the travel time from a location $u \in V$ to a location $v \in V$ (as well as the arrival time at v) depends on the leave time from u and the chosen transportation mode (e.g on foot or public transportation).

In TDTOPTW we assume that the starting and ending locations may be different for different routes. Therefore, $s_r, t_r \in V$ denote the starting, terminal location respectively of the $r-$th route, and st_r, et_r denote the starting, ending time respectively of the $r-$th route, $r = 1, 2, \ldots, m$.

The proposed TDCSCRoutes and SlackCSCRoutes algorithms modify the CSCRoutes algorithm for TOPTW [11] to handle time dependent travel times among different locations/POIs. CSCRoutes is a cluster-based heuristic that achieves best performance results with respect to execution time compared to the best known so far real-time TOPTW algorithm, ILS [10].

The algorithms introduced in this section, employ an insertion step which takes into account the fact that for each pair of locations u and v the travel time from u to v may vary (a tourist can choose between walking and using public transport), and the waiting time for public transport depends on the time the tourist arrives at u. In Subsection 3.1 we present the feasibility criterion for inserting a POI p in a route r in the case of time dependent travel costs.

3.1 Time dependent insertion feasibility

In order to have the time dependent travel cost between all pairs of locations, for each (u, v), $u, v \in V$ we precalculate the walking time from u to v (might be ∞, when too far to walk) and a set S_{uv} containing schedule information of the public transportation system connecting u and v. Specifically, S_{uv} contains all the non-dominated pairs $(\text{dep}_i^{uv}, \text{trav}_i^{uv}), i = 1, 2, \ldots, |S_{uv}|$ in ascending order of dep_i^{uv}, where dep_i^{uv} is a departure time and trav_i^{uv} is the corresponding travel time of a service of public transport connecting u and v. We consider that a pair $(\text{dep}_i^{uv}, \text{trav}_i^{uv})$ dominates a pair $(\text{dep}_j^{uv}, \text{trav}_j^{uv})$ if $\text{dep}_i^{uv} + \text{trav}_i^{uv} \leq \text{dep}_j^{uv} + \text{trav}_j^{uv}$ and $\text{dep}_i^{uv} > \text{dep}_j^{uv}$. Note that departing from u at time t where $\text{dep}_i^{uv} < t \leq \text{dep}_{i+1}^{uv}$, will result in arriving at v either at the same time as if departing at dep_{i+1}^{uv}, or at time t plus the walking time from u to v. More specifically, the arrival time at v will be equal to the earliest of the times $\text{dep}_{i+1}^{uv} + \text{trav}_{i+1}^{uv}$ and $t + \text{walking}_{u,v}$, where $\text{walking}_{u,v}$ is the walking time from u to v.

For a specified time t, the departure time from u to v at t using public transport, $\text{deptime}_{u,v}(t)$, is defined as the earliest possible departure time from u to v, i.e.,

$$\text{deptime}_{u,v}(t) = \min_i \{\text{dep}_i^{uv} | (\text{dep}_i^{uv}, \text{trav}_i^{uv}) \in S_{uv} \text{ and } t \leq \text{dep}_i^{uv}\} \quad (1)$$

Then, the travel time from u to v at t using public transport, $\text{travtime}_{u,v}(t)$, is such that $(\text{deptime}_{u,v}(t), \text{travtime}_{u,v}(t)) \in S^{uv}$, and the departure delay at time t due to the use of public transport, is $\text{delay}_{u,v}(t) = \text{deptime}_{u,v}(t) - t$. Therefore, the total travelling cost from u to v at a specified time t, $\text{travelling}_{u,v}(t)$, is

$$\text{travelling}_{u,v}(t) = \min\{\text{walking}_{u,v}, \text{delay}_{u,v}(t) + \text{travtime}_{u,v}(t)\} \quad (2)$$

For a POI p_i in a route r the following variables are defined:

- wait_i, denoting the waiting time at p_i before its time window starts; $\text{wait}_i = \max(0, \text{open}_{ir} - \text{arrive}_i)$.
- start_i, denoting the starting time of the visit at p_i; $\text{start}_i = \text{arrive}_i + \text{wait}_i$.
- leave_i, denoting the time the visit at p_i completes, i.e., the departure time from p_i; $\text{leave}_i = \text{start}_i + \text{visit}_i$.
- arrive_i, denoting the arrival time at p_i; $\text{arrive}_i = \text{leave}_{\text{prev}(i)} + \text{travelling}_{\text{prev}(i),i}(\text{leave}_{\text{prev}(i)})$, where $\text{leave}_{\text{prev}(i)}$ is the departure time from the previous node of p_i in route r ($\text{prev}(i)$). We assume that $\text{arrive}_{s_r} = st_r$.
- maxStart_i, denoting the latest time the visit at p_i can start without violating the time windows of the nodes following p_i; $\text{maxStart}_i = \min(\text{close}_{ir}, \max\{t : t + \text{travelling}_{i,\text{next}(i)}(t) \leq \text{maxStart}_{\text{next}(i)}\} - \text{visit}_i)$, where $\text{next}(i)$ is the node following p_i in r. We assume that $\text{maxStart}_{t_r} = et_r$.

A POI p_k can be inserted in route r between POIs p_i and p_j if the arrival time at p_k does not violate p_k's time window and the arrival at p_j does not violate the time window of p_j as well as the time windows of the nodes following p_j in r. The total time cost for p_k's insertion is defined as shift_k^{ij} (insertion cost) and is equal to the time the arrival at p_j will be delayed. In particular shift_k^{ij} equals to the time required to travel from p_i to p_j having visited p_k in between minus the time taken for travelling directly from p_i to p_j.

$$\text{shift}_k^{ij} = (\text{travelling}_{i,k}(\text{leave}_i) + \text{wait}_k + \text{visit}_k + \text{travelling}_{k,j}(\text{leave}_k)) - \text{travelling}_{i,j}(\text{leave}_i) \tag{3}$$

Figure 1 illustrates an example of inserting p_k, between p_i and p_j shifting the visit at p_j later on time (in this figure, wait_u^v denotes the waiting at u following a visit at v).

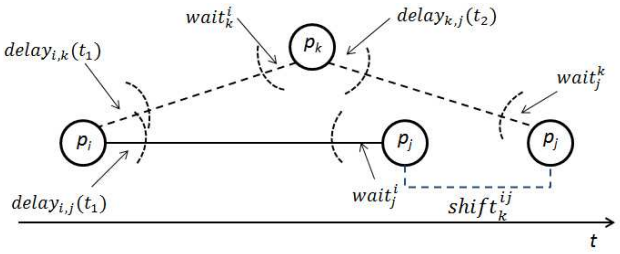

Fig. 1. Illustration of p_k insertion between p_i and p_j.

Note that the insertion of p_k between p_i and p_j in route r is feasible when

$$\text{arrive}_k \leq \text{close}_{kr} \text{ and } \text{shift}_k^{ij} \leq \text{maxStart}_j - \text{arrive}_j \tag{4}$$

A pseudo code implementation of the function $\text{shift}(k, i, j, r)$ which calculates the insertion cost shift_k^{ij} in route r, in given in [15]. The function returns ∞ if the insertion of p_k is infeasible.

3.2 The Time Dependent CSCRoutes (TDCSCRoutes) algorithm

TDCSCRoutes algorithm modifies the insertion step **CSCRoutes_Insert** of CSCRoutes algorithm to handle time dependent travel times among different locations/POIs. CSCRoutes uses the notion of *Cluster Route (CR)* defined as follows: Given a route r of a TOPTW solution, any maximal sub-route in r comprising a sequence of nodes within the same cluster C is called a *Cluster Route (CR)* of r *associated with cluster* C and denoted as CR^r_C. CSCRoutes algorithm is designed to construct routes that visit each cluster at most once, i.e. if a cluster C has been visited in a route r it cannot be revisited in the same route and therefore, for each cluster C there is only one cluster route in any route r associated with C. The only exception allowed is when the start and the terminal nodes of a route r belong to the same cluster C'. In this case, a route r may start and end with nodes of cluster C', i.e. C' may be visited twice in the route r and therefore, for a route r there might be two cluster routes $CR^r_{C'}$. The insertion step **CSCRoutes_Insert** of CSCRoutes does not allow the insertion of a POI p_k in a route r, if this insertion creates more than one cluster routes CR^r_C for some cluster C. Therefore, a POI cannot be inserted at any position in the route r [11].

In the sequel, the description of the insertion step of TDCSCRoutes (**TDCS Routes_Insert**) is given. It comprises a modification of **CSCRoutes_Insert** which takes into consideration the time dependent travel times among locations/POIs. Given a route r let CR^r_f be the first cluster route (starting at s_r) in r, and CR^r_l be the last cluster route (ends at t_r) in r. Let also clustersIn(r) be a set containing any cluster C for which there is a nonempty CR^r_C, and cluster(p) be the cluster where p belongs to. Given a candidate for insertion POI p_k **TDCSCRoutes_Insert** distinguishes among the following cases:

- cluster(s_r) = cluster(t_r)
  - if clustersIn(r) = {cluster(s_r)} then p_k can be inserted anywhere in the route.
  - if clustersIn(r) $\neq$ {cluster(s_r)} and cluster(p_k) = cluster(s_r) then p_k can be inserted in CR^r_f and CR^r_l
  - if clustersIn(r) $\neq$ {cluster(s_r)} and cluster(p_k) $\neq$ cluster(s_r) and cluster(p_k) $\notin$ clustersIn(r) then p_k can be inserted after every end of a CR except for CR^r_l
  - if clustersIn(r) $\neq$ {cluster(s_r)} and cluster(p_k) $\neq$ cluster(s_r) and cluster(p_k) $\in$ clustersIn(r) then p_k can be inserted anywhere in $CR^r_{\text{cluster}(p_k)}$
- cluster(s_r) $\neq$ cluster(t_r)
  - if cluster(p_k) = cluster(s_r) then p_k can be inserted everywhere in CR^r_f
  - if cluster(p_k) = cluster(t_r) then p_k can be inserted everywhere in CR^r_l
  - if cluster(p_k) $\in$ clustersIn(r) and cluster(p_k) is different from cluster(s_r) and cluster(t_r), then p_k can be inserted everywhere in $CR^r_{\text{cluster}(p)}$
  - if cluster(p_k) $\notin$ clustersIn(r) then p_k can be inserted at the end of any CR in r except for CR^r_l

For each POI p_k not included in a route, among all feasible insert positions (between POIs p_i, p_j) we select the one with the highest ratio

$$\text{ratio}_k^{ij} = \frac{\text{profit}_k^2}{\text{shift}_k^{ij}}(1 + a \cdot \frac{D_k^{ij} + 1}{D_k^{ij} + 2} + (1 - a) \cdot f(\text{shift}_k^{ij}, \text{wait}_j + \text{delay}_j)) \quad (5)$$

where $f(x, y) = 1$ if $x \leq y$ and 0 otherwise, and $D_k^{ij} = \text{delay}_{i,k}(\text{leave}_i) + \text{wait}_k + \text{delay}_{k,j}(\text{leave}_k) + \text{wait}_j$ where a takes the values of $1, \frac{1}{2}$ and 0, depending on the number of iterations executed by CSCRoutes. In particular, for the first $\frac{1}{3}$ iterations a is equal to 1, it decreases to $\frac{1}{2}$ in the second $\frac{1}{3}$ iterations and becomes 0 in the final iterations [11]. The incentive behind (5) is the following: $\frac{\text{profit}_k^2}{\text{shift}_k^{ij}}$ denotes preference for important (i.e. highly profitable) POIs associated with relatively short time to visit. In the first iterations (a=1), the operand $\frac{D_k^{ij}+1}{D_k^{ij}+2}$ dominates giving preference to insertion of POIs among pairs (p_i, p_j) creating prolonged 'empty' time periods (i.e. long aggregate waiting times and delays) to be utilized on later insertions. In the last iterations (a=0), $f(\text{shift}_k^{ij}, \text{wait}_j + \text{delay}_j)$ dominates favoring insertion of POIs that best take advantage of any left unexploited time (i.e. waiting and delays) remaining throughout the routes. Among all candidate POIs, TDCSCRoutes algorithm selects for insertion the one associated with the highest ratio.

Once a POI p_k is inserted between p_i and p_j in a route r, the variable values of all POIs in r need to be updated. Note that for each POI after p_k, the variables arrive, wait, start and leave should be updated while variable maxStart remains the same. For each POI p_l before p_k the value of maxStart_l is the only one that should be updated . The pseudo code of **TDCSCRoutes_Insert** is given in [15].

3.3 The SlackCSCRoutes algorithm

SlackCSCRoutes modifies the insertion step of TDCSCRoutes i.e., it follows a different approach for determining the POI p_k that will be selected for insertion in a route r. Specifically, while TDCSCRoutes algorithm's criterion for selecting a POI p_k in a route r is based on the insertion cost, SlackCSCRoutes involves a more global criterion as it takes into consideration the effect of this insertion in the whole route r.

SlackCSCRoutes uses an additional variable slack_i defined for each node p_i in a tourist route r as follows:

$$\text{slack}_i = \text{maxStart}_i - \text{arrive}_i \quad (6)$$

Note that if the value of slack_i is close to 0 then there is little hope in finding new POIs that can be inserted between POIs $p_{\text{prev}(i)}$ and p_i.

Let $p_1, p_2, \ldots, p_n$ be the successive POIs of a route r with $p_1 = s_r$ and $p_n = t_r$. Let p_k be a candidate POI for insertion between POIs p_i and p_{i+1} of r.

The insertion of the p_k will likely shift further the arrival time at p_j (arrive_j), for $j = i+1, \ldots, n$. That depends on the waiting time before the visit of each POI and the time dependent travelling time for moving between successive nodes along the route. Let arrive_j^k be the new arrival time at POI p_j after the insertion of p_k, for $j = i+1, \ldots, n$, . The above insertion may shift the maximum time the visit at p_j can start (maxStart_j) ahead for $j = 1, \ldots, i$. Let maxStart_j^k be the new latest time the visit at p_j can start after the insertion of p_k, for $j = 1, \ldots, i$.

Let also $\text{slack}_j^k = \text{maxStart}_j - \text{arrive}_j^k$, for $j = i+1, \ldots, n$, and $\text{slack}_j^k = \text{maxStart}_j^k - \text{arrive}_j$, for $j = 1, \ldots, i$, be the corresponding values of the "slack" variables. We define the quantity A_k^i as follows:

$$A_k^i = \frac{\sum_{j=1}^{i} \text{slack}_j^k + \text{slack}_k + \sum_{j=i+1}^{n} \text{slack}_j^k}{n+1}$$

Note that a large value of A_k^i implies that even after the insertion of p_k, there are many possibilities left for inserting new POIs along each leg of trip (that is, prior and after visiting p_k).

Then for each POI p_k, the maximum possible A_k^i is determined, i.e. the best possible insert position. Let the maximum value A_k^i over all possible insert positions be A_k. Then, in order to determine the POI that will be selected for insertion, the slackWeight for each POI p_k is calculated as

$$\text{slackWeight}_k = {\text{profit}_k}^2 * A_k$$

and the POI with the highest slackWeight is inserted.

The main issue with the above derivations is that for each POI p_k and for each possible insert position i within a route r we need to calculate A_k^i which involves the updated values of the maxStart and arrive variables for all POIs in r. This involves a global rather than a local decision perspective regarding possible insertion positions along the whole route. In order to develop a fast heuristic, a quick calculation of A_k^i is necessary. We may have a quick calculation of a good approximation of A_k^i, by making the assumption that the time windows at the POIs are fairly long spanning the most part of the day and therefore the waiting time before each POI is typically zero (Details will be given in the full version of the paper). Note this assumption is realistic for most tourist sites.

4 Experimental Results

4.1 Test Instances

While many different datasets exist for testing (T)OP(TW) problems, this is not the case for their time-dependent counterparts. Hence, relevant algorithmic solutions should unavoidably be tested upon real transit network data. In our experiments, we have used the GTFS (General Transit Feed Specification) data of the transit network deployed on the metropolitan area of Athens, Greece, provided by the Athens Urban Transport Organization. The network comprises

3 subway lines, 3 tram lines and 287 bus lines with an overall of 7825 transit stops. For our purposes, we require to know the pairwise quickest routes full (24h range) multimodal travel times between POIs, for all possible departure times of the day. This precomputation has been performed using the the algorithm of Dibbelt et al. [16] upon the Athens transit network. The overall shortest time dependent travel time information (either through transit or walking) 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. We have also used a set of predefined start/end locations (100 hotels).

The POIs dataset used in our experiments features 113 sites (museums, archaeological sites, landmarks, streets & squares, neighborhoods, religious heritage, parks) mostly situated around Athens downtown and Piraeus areas. Profits have been set in a 1-100 scale and visiting times vary from 1 minute (e.g. for some outdoor statues) to 2 hours (e.g. for some not-miss museums and wide-area archaeological sites). The POIs have been grouped in $\lfloor \frac{N}{10} \rfloor = 11$ disjoint clusters.

The above described POIs dataset has been used to create three different 'topologies'. The real POIs coordinates have been maintained in all cases, however, their respective profits, visiting times and opening hours (i.e. time windows) have been 'shuffled', to remove any potential bias of a single topology.

Our algorithms have been tested using 100 different 'user preference' inputs per number of routes, each applied to all the three abovementioned topologies. Each 'preference' input is associated with a different start/end location, corresponding to a potential accommodation (hotel) option. Furthermore, for each 'preference' input a certain percentage of POIs is disregarded to 'simulate' preferences provided by real visitors, such as no interest on religious sites. The total time budget available for sightseeing in daily basis (B_r) has been set to 5 hours (10:00-15:00) in all experiments. All test instances-related files are accessible from: `http://www2.aegean.gr/dgavalas/public/tdtoptw_instances/index.html`

4.2 Results

We have implemented the following four algorithms: (a) TDCSCRoutes (see Section 3.2), (b) SlackCSCRoutes (see Section 3.3), (c) AvgCSCRoutes, and (d) Average ILS (AvgILS).

AvgILS refers to the average travel time approach proposed by Garcia et al. [1], wherein TDTOPTW is practically reduced to TOPTW and the standard ILS algorithm [10] is used to construct routes based on pre-computed average travel times. AvgILS exercises a repair procedure, introducing the real travel times between the POIs of the final TOPTW solution. If this causes a visit to become infeasible, the latter is removed from the route and the remainder of the route is shifted forward. Similarly to AvgILS, AvgCSCRoutes uses CSCRoutes to construct routes based on pre-computed average travel times. AvgCSCRoutes employs a repair step similar to AvgILS, followed by a 'gap filling' step the

latter inserts new POIs into the routes, if feasible, thereby further improving the solution's quality (Details will be given in the full version of the paper).

All algorithms have been employed upon the test instances described in the previous subsection, deriving k daily personalized routes, $k = 1 \dots 4$, each for every day of stay at the destination. All routes start and end at the tourist's accommodation location. Note that all the algorithms have been programmed in C++ and executed on a PC Intel Core i5, clocked at 2.80GHz, with 4GB RAM.

We use the standard profit criterion in our experimental results , i.e. we consider as best-found solution the one with the highest aggregate profit. Table 1 offers a comparative view on the performance of the four implemented algorithms. The analytical results may be found in [15]. The results shown are: the overall collected profit (over all routes); the execution time (in ms); the number of visited POIs; the overall number of public transit transfers (PT) over all routes; All the above results are averaged over all (= 100) the execution runs and the three 'shuffled' topologies. High quality solutions are those featuring high aggregate profit and relatively small number of transit transfers, derived in short execution time. Note that we normalize the actual performance parameter values assigning a value 100 to the highest recorded value and adapting the rest accordingly (this allows illustrating relative performance gaps among tested algorithms). Performance values shown in bold designate the best performing algorithm with respect to each performance parameter.

As shown in Table 1, AvgCSCRoutes executes considerably faster, since it disregards time dependency on the insertion decision, while also using a smaller memory structure to hold travel time information (hence, required travel times are retrieved more efficiently). Despite using precomputed average travel time values, AvgILS executes (on average) slower than the other three algorithms as it explores a larger search space on each POI insertion. It should be noted though that execution times are well beyond a second in all cases for all algorithms for the 113 POIs of Athens.

Interestingly, AvgCSCRoutes and AvgILS are competitive in terms of profit, with AvgCSCRoutes performing better than AvgILS, mainly due to the extra 'gap filling' step, which considerably improves the quality of its solutions and corrects potential suboptimal node insertion decisions made during the insertion phase. Nevertheless, we argue that the results obtained by AvgCSCRoutes and AvgILS could be worse when considering either less frequent transit services or timetables where transit frequencies changes considerably along the day or even when considering tourist visits in off-peak hours. In such scenarios, using the average travel time would not serve as a good approximation.

TDCSCRoutes performs marginally better than the three other algorithms with respect to the overall profit, while deriving solutions in comparable time with SlackCSCRoutes.

On the other hand, TDCSCRoutes performs worse in terms of public transit transfers, mainly due to its initialization phase which favors visits to clusters far located from the user's accommodation, hence, often requiring public transit rides to arrive there. AvgILS obtains best results with respect to that perfor-

mance metric. Last, SlackCSCRoutes achieves higher number of POI visits (intuitively, due to its insertion criterion, the algorithm best exploits the available time budget, accommodating more POI visits).

Table 1. Comparative view on the performance of the implemented algorithms

# Routes		Profit	Time	Visits	Public Transport
1	TDCSCRoutes	**100**	89.88	97.57	100
	SlackCSCRoutes	99.99	100	**100**	92.92
	AvgCSCRoutes	98.38	**55.98**	96.41	88.78
	AvgILS	96.68	79.64	92.72	**85.84**
2	TDCSCRoutes	**100**	87.21	98.09	100
	SlackCSCRoutes	99.18	93.75	**100**	95.42
	AvgCSCRoutes	98.82	**51.07**	97.73	88.13
	AvgILS	97.64	100	95.14	**85**
3	TDCSCRoutes	**100**	92.05	97.1	100
	SlackCSCRoutes	98.15	89.01	**100**	90.48
	AvgCSCRoutes	99.05	**50.39**	97.39	80.26
	AvgILS	98.05	100	95.03	**78.77**
4	TDCSCRoutes	**100**	100	96.32	100
	SlackCSCRoutes	97.88	87.11	**100**	93.81
	AvgCSCRoutes	99.23	**52.34**	96.42	82.71
	AvgILS	98.21	88.72	94.03	**82.38**

5 Conclusions and Future Work

We introduced TDCSCRoutes and SlackCSCroutes, two new cluster-based heuristics for solving the TDTOPTW. The main design objectives of the two algorithms are to derive high quality TDTOPTW solutions (maximizing tourist satisfaction), while minimizing the number of transit transfers and executing fast enough to support online web and mobile applications.

With respect to the overall collected profit, TDCSCRoutes has been shown to perform marginally better. On the other hand, SlackCSCRoutes achieves a fair compromise among all the performance aspects. In practical applications, comprising very large datasets, AvgCSCRoutes could be the most suitable choice as it efficiently derives solutions of reasonably good quality (this conclusion agrees with that reported in [1], wherein a TDTOPTW algorithm has been evaluated against AvgILS). Nevertheless, its suitability largely depends on the high frequency of public transit services, so that average travel times represent a good guess.

In the future, we plan to test our algorithms on additional real datasets to remove potential bias introduced by the particularities of the Athens dataset and transit network. Besides, testing our algorithms over larger POI datasets will verify their scalability in terms of the required execution time. Along the

same line, we plan to produce realistic synthesized multimodal timetabled data to serve as additional test benchmarks.

References

1. Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M.T.: Integrating public transportation in personalised electronic tourist guides. Computers & Operations Research **40**(3) (2013) 758 – 774
2. Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. European Journal of Operational Research **209**(1) (2011) 1 – 10
3. Laporte, G., Martello, S.: The selective travelling salesman problem. Discrete Applied Mathematics **26**(2-3) (1990) 193 – 207
4. Gambardella, L., Montemanni, R., Weyland, D.: Coupling ant colony systems with strong local searches. European Journal of Operational Research **220**(3) (2012) 831 – 843
5. Labadi, N., Melechovský, J., Wolfler Calvo, R.: Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics **17** (2011) 729–753
6. Labadi, N., Mansini, R., Melechovský, J., Wolfler Calvo, R.: The team orienteering problem with time windows: An lp-based granular variable neighborhood search. European Journal of Operational Research **220**(1) (2012) 15 – 27
7. Lin, S.W., Yu, V.F.: A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research **217**(1) (2012) 94 – 107
8. Montemanni, R., Gambardella, L.M.: An ant colony system for team orienteering problems with time windows. Foundations of Computing and Decision Sciences **34**(4) (2009) 287–306
9. Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F.: Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research **37**(2) (2010) 351 – 367
10. Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Computers & Operations Research **36** (2009) 3281–3290
11. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Tasoulas, Y.: Cluster-based heuristics for the team orienteering problem with time windows. In: Proceedings of 12th International Symposium on Experimental Algorithms (SEA'13). (2013) 390–401
12. Fomin, F.V., Lingas, A.: Approximation algorithms for time-dependent orienteering. Information Processing Letters **83**(2) (2002) 57 – 62
13. Spieksma, F.C.R.: On the approximability of an interval scheduling problem. Journal of Scheduling **2** (1999) 215–227
14. Abbaspour, R.A., Samadzadegan, F.: Time-dependent personal tour planning and scheduling in metropolises. Expert Systems and Applications **38** (2011) 12439–12452
15. Gavalas, D., et al.: Appendix. `http://www2.aegean.gr/dgavalas/public/tdtoptw_instances/icaa_appendix.pdf`
16. Dibbelt, J., Pajor, T., Wagner, D.: User-constrained multi-modal route planning. In: Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12). (2012) 118 – 129