Cluster-based Heuristics for the Team Orienteering Problem with Time Windows

Damianos Gavalas[1], Charalampos Konstantopoulos[2], Konstantinos Mastakas[3], Grammati Pantziou[4], and Yiannis Tasoulas[2]

[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,jtas}@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

Abstract. The Team Orienteering Problem with Time Windows (TOPTW) deals with deriving a number of tours comprising a subset of candidate nodes (each associated with a "profit" value and a visiting time window) so as to maximize the overall "profit", while respecting a specified time span. TOPTW has been used as a reference model for the Tourist Trip Design Problem (TTDP) in order to derive near-optimal multiple-day tours for tourists visiting a destination featuring several points of interest (POIs), taking into account a multitude of POI attributes. TOPTW is an NP-hard problem and the most efficient known heuristic is based on Iterated Local Search (ILS). However, ILS treats each POI separately; hence it tends to overlook highly profitable areas of POIs situated far from the current location, considering them too time-expensive to visit. We propose two cluster-based extensions to ILS addressing the aforementioned weakness by grouping POIs on disjoint clusters (based on geographical criteria), thereby making visits to such POIs more attractive. Our approaches improve on ILS with respect to solutions quality, while executing at comparable time and reducing the frequency of overly long transfers among POIs.

Keywords: Tourist Trip Design Problem, Point of Interest, Team Orienteering Problem with Time Windows, Iterated Local Search, Clustering.

1 Introduction

A TTDP [15] refers to a route-planning problem for tourists interested in visiting multiple points of interest (POIs). The objective of the TTDP is to select POIs that match tourist preferences, thereby maximizing tourist satisfaction, while taking into account a multitude of parameters and constraints (e.g., distances among POIs, visiting time required for each POI, POIs visiting hours, entrance fees) and respecting the time available for sightseeing in daily basis. Different versions of TTDP have been studied in the literature. Herein, we deal with a version of TTDP that considers the following input data: (a) a set of candidate POIs, each associated with the following attributes: a location (i.e. geographical coordinates), time windows(TW) (i.e. opening hours), a "profit" value, calculated as a weighted function of the objective and subjective importance of the POI (subjectivity refers to the users' individual preferences and interests on specific POI categories) and a visiting time (i.e. the anticipated duration of visit of a user at the POI), (b) the travel time among POIs, based on the topological distance between a pair of POIs, (c) the number k of routes that must be generated, based upon the period of stay (number of days) of the tourist at the destination, and (d) the daily time budget B that a tourist wishes to spend on visiting sights; the overall daily route duration (i.e. the sum of visiting times plus the overall travel time among visited POIs) should be kept below B.

By solving the TTDP we expect to derive k routes (typically starting and ending at the tourist's accommodation location) each of length at most B, that maximize the overall collected profit. A well-known optimization problem that may formulate this version of TTDP is the team orienteering problem with time windows (TOPTW) [13]. TOPTW is NP-hard (e.g. see [3], [6]). Hence, exact solutions for TOPTW are feasible for instances with very restricted number of locations (e.g. see the work of Li and Hu [7], which is tested on networks of up to 30 nodes). Note that since the TTDP is typically dealt with online web and mobile applications with strict execution time restrictions, only highly efficient heuristic approaches are eligible for solving it. The most efficient known heuristic for TOPTW is based on Iterated Local Search (ILS) [14], offering a fair compromise with respect to execution time versus deriving routes of reasonable quality. 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. ILS is also often trapped in areas with isolated high-profit POIs, possibly leaving considerable amount of the overall time budget unused.

Herein, we introduce CSCRatio and CSCRoutes, two cluster-based algorithmic approaches to the TTDP, which address the shortcomings of ILS. The main incentive behind our approaches is to motivate visits to topology areas featuring high density of 'good' candidate nodes (such areas are identified by a geographical clustering method performed offline); the aim is to improve the quality of derived solutions while not sacrificing time efficiency. Furthermore, both our algorithms favor solutions with reduced number of overly long transfers among nodes, which typically require public transportation rides (such transfers are costly and usually less attractive to tourists than short walking transfers). The remainder of this article is organized as follows: Section 2 overviews TOPTW heuristics. Section 3 presents our novel cluster-based heuristics, while Section 4 discusses the experimental results. Section 5 concludes the paper.

2 Related work

Labadi et al. [4] proposed a local search heuristic algorithm for TOPTW based on a variable neighborhood structure. In the local search routine the algorithm tries to replace a segment of a path by nodes offering more profit. For that, an assignment problem related to the TOPTW is solved and based on that solution the algorithm decides which arcs to select. Lin et al. [9] proposed a heuristic algorithm for TOPTW based on simulated annealing. On each iteration a neighbouring solution is obtained from the current solution by applying one of the moves swap, insertion or inversion, with equal probability. A new solution is adopted provided that it is more profitable than the current one; otherwise, the new solution might again replace the current one with a probability inversely proportional to their difference in profits. After applying the above procedure for a certain number of iterations the best solution found so far is further improved by applying local search.

The Iterated Local Search (ILS) heuristic proposed by Vansteenwegen et al. [14] is the fastest known algorithm proposed for TOPTW [13]. The algorithm is discussed in the following section. Montemanni and Gambardella proposed an ant colony system (ACS) algorithm [10] to derive solutions for a hierarchical generalization of TOPTW, wherein more than the k required routes are constructed. At the expense of the additional overhead, those additional fragments are used to perform exchanges/insertions so as to improve the quality of the k tours. ACS has been shown to obtain high quality results (that is, low average gap to the best known solution) at the expense of prolonged execution time, practically prohibitive for online applications. In [2] a modified ACS framework (Enhanced ACS) is presented and implemented for the TOPTW to improve the results of ACS.

Labadi et al. [5] recently proposed a method that combines the greedy randomized adaptive search procedure (GRASP) with the evolutionary local search (ELS). Their approach derives solutions of comparable quality and significantly less computational effort to ACS. Compared to ILS, GRASP-ELS gives better quality solutions at the expense of increased computational effort [5]. Tricoire et al. [12] deal with the Multi-Period Orienteering Problem with Multiple Time Windows (MuPOPTW), a generalization of TOPTW, wherein each node may be assigned more than one time window on a given day, while time windows may differ on different days. Both mandatory and optional visits are considered. The authors developed two heuristic algorithms for the MuPOPTW: a deterministic constructive heuristic which provides a starting solution, and a stochastic local search algorithm, the Variable Neighbourhood Search (VNS), which considers random exchanges between chains of nodes. Vansteenwegen et al. [13] argue that a detailed comparison of ILS, ACS and the algorithm of Tricoire et al. [12], is impossible since the respective authors have used (slightly) different benchmark instances. Nevertheless, it can be concluded that ILS has the advantage of being very fast (its execution time is no longer than a few seconds), while ACS, Enhanced ACS and the approach of Tricoire et al. have the advantage of obtaining high quality solutions.

3 Cluster-based heuristics

In TOPTW we are given a directed graph $G = (V, A)$ where $V = \{1, ..., N\}$ is the set of nodes (POIs) and A is the set of links, an integer k, and a time budget B. The main attributes of each node are: the service or visiting time (visit_i), the profit gained by visiting i (profit_i), and each day's time window ($[\text{open}_{im}, \text{close}_{im}], m = 1, 2, \dots, k$) (a POI may have different time windows per day). Every link $(i, j) \in A$ denotes the transportation link from i to j and is assigned a travel cost travel_{ij}. The objective is to find k disjoint routes starting from 1 and ending at N, each with overall duration limited by the time budget B, that maximize the overall profit collected by visited POIs in all routes.

The ILS heuristic proposed by Vansteenwegen et al. [14] defines an "insertion" and a "shake" step. At each insertion step (**ILS_Insert**) a node is inserted in a route, ensuring that all following nodes in the route remain feasible to visit, i.e. their time window constraints are satisfied and the time budget is not violated. ILS modeling involves two additional variables for each node i: (a) wait_i defined as the waiting time in case the arrival at i takes place before i's opening time, and (b) maxShift_i defined as the maximum time the start of the visit of i can be delayed without making any visit of a POI in the route infeasible. If a node p is inserted in a route t between i and j, let $\text{shift}_p = \text{travel}_{ip} + \text{wait}_p + \text{visit}_p + \text{travel}_{pj} - \text{travel}_{ij}$ denote the time cost added to the overall route time due to the insertion of p. The node p can be inserted in a route t between i and j if and only if $\text{start}_{it} + \text{visit}_i + \text{travel}_{ip} \leq close_{pt}$ and at the same time $\text{shift}_p \leq \text{wait}_j + \text{maxShift}_j$. For each node p not included in a route, its best possible insert position is determined by computing the lowest insertion time cost (shift_p). For each of these possible insertions the heuristic calculates the ratio $\text{ratio}_p = \frac{\text{profit}_p^2}{\text{shift}_p}$, which represents a measure of how profitable is to visit p versus the time delay this visit incurs. Among all candidate nodes, the heuristic selects for insertion the one with the highest ratio .

At the shake step (**Shake**) the algorithm tries to escape from local optimum by removing a number of nodes in each route of the current solution, in search of non-included nodes that may either decrease the route time length or increase the overall collected profit. The shake step takes as input two integers: (a) the removeNumber that determines the number of the consecutive nodes to be removed from each route and (b) the startNumber that indicates where to start removing nodes on each route of the current solution. If throughout the process, the end location is reached, then the removal continues with the nodes following the start location.

To the best of our knowledge, ILS is the fastest known algorithm for solving the TOPTW offering a fair compromise in terms of speed versus deriving routes of reasonable quality. However, it presents the following weaknesses: (i) During the insertion step, ILS may rule out candidate nodes with high profit value because they are relatively time-expensive to reach (from nodes already included in routes). This is also the case even when whole groups of high profit nodes are located within a restricted area of the plane but far from the current route instance. In case that the route instance gradually grows and converges towards the high profit nodes, those may be no longer feasible to insert due to overall route time constraints. (ii) In the insertion step, ILS may be attracted and include into the solution some high-score nodes isolated from high-density topology areas. This may trap ILS and make it infeasible to visit far located areas with "good" candidate nodes due to prohibitively large traveling time (possibly leaving considerable amount of the overall time budget unused).

Herein, we propose two heuristic algorithms, Cluster Search Cluster Ratio (CSCRatio) and Cluster Search Cluster Routes (CSCRoutes), which address the aforementioned weaknesses of the ILS algorithm. Both algorithms employ clustering to organize POIs into groups (clusters) based on topological distance criteria. POIs at the same cluster are close to each other e.g., they are within walking distance or they belong to the same area of the city. Having visited a high-profit POI that belongs to a certain cluster, our algorithms encourage visits to other POIs at the same cluster because such visits reduce (a) the duration of the routes and (b) the number of transfers among clusters. Note that a tourist apart from maximizing the total profit, may also prefer to minimize inter-cluster tranfers as those are typically long and require usage of public transportation, this may incur a considerable budget cost, while walking is usually a preferred option than using the public transportation.

Both CSCRatio and CSCRoutes employ the global k-means algorithm [8] to build a clustering structure consisting of an appropriate (based on the network topology) number of clusters (numberOfClusters). Once the clusters of POIs have been formed during a preprocessing (clustering) phase, a route initialization phase **RouteInitPhase** starts. During this phase one POI is inserted into each of the k initially empty routes. Each of the k inserted POIs comes from a different cluster, i.e. no two inserted POIs belong to the same cluster. Since the number of clusters is usually larger that k we need to decide which k clusters will be chosen in the route initialization phase. Different approaches may be followed such as choosing the k clusters with the highest total profit, or trying different sets of k high-profit clusters and run CSCRatio and CSCRoutes algorithms for each such set searching for the best possible solution. Following the second approach, we consider a listOfClusterSets list containing a specific number of different sets of k high-profit clusters. The list may contain all k-combinations of the elements of a small set S with the most profitable clusters. **RouteInitPhase** takes as argument a set of k clusters from listOfClusterSet and proceeds as follows: for each cluster C_i in the set, it finds the POI $p \in C_i$ with the highest ratio$_p$ and inserts it into one of the empty routes. By initializing each one of the k routes of the TOPTW solution with a POI from different clusters the algorithms encourage searching different areas of the network and avoid getting trapped at specific high-scored nodes. Then the algorithms combine an insertion step and a shake step to escape from local optima.

Cluster Search Cluster Ratio Algorithm. The CSCRatio algorithm introduces an insertion step **CSCRatio_Insert** which takes into account the clustering of the POIs by using a parameter clusterParameter ≥ 1. The higher the value of clusterParameter, the more the insertion of a node p before or after a node that belongs to the same cluster with p is favored. Specifically, the parameter clusterParameter is used to increase the likelihood of inserting p between i and j if p belongs to the same cluster with either i or j. For that, CSCRatio considers the variable shiftCluster_p defined as the ratio $\frac{\text{shift}_p}{\text{clusterParameter}}$ in the case that cluster(p) coincides with cluster(i) or cluster(j) (cluster(l) denotes the cluster where a node l belongs to). Otherwise, $\text{shiftCluster}_p = \text{shift}_p$. Then the lowest insertion time cost (shiftCluster_p) i.e. the best possible insert position for p, is determined. For each of those best possible insertions, the heuristic calculates $\text{ratio}_p = \frac{\text{profit}_p^2}{\text{shiftCluster}_p}$. CSCRatio initializes the clusterParameter with the value of 1.3 in order to initially encourage visits to be within the same clusters and decreases the value of clusterParameter by 0.1 every a quarter of maxIterations. At the last quarter the **CSCRatio_Insert** step becomes the same as **ILS_Insert**. Thus, routes with a lot of POIs belonging to the same cluster are initially favored, while as the number of iterations without improvement increases, the diversification given by ILS is obtained.

The maximum value of the parameter removeNumber used in the shake step is allowed to be half of the size of the largest route (currentSolution.maxSize) in the current solution and not $\frac{N}{3k}$ as in ILS [14]. In this way, execution time is saved, since local optimum is reached in short time, if a small portion of the solution has been removed. As a result, the number of iterations of CSCRatio can be larger than the number of iterations of ILS [14] without increasing the overall algorithm's execution time.

CSCRatio loops for a number of times equal to the size of the listOfClusterSets. Within the loop, firstly all POIs included into the current solution's routes are removed and the route initialization phase is executed with argument a set of high-profit clusters taken (pop operation) from the listOfClusterSets list. Secondly, the algorithm initializes the parameters startNumber and removeNumber of **Shake** to 1 and the parameter clusterParameter of **CSCRatio_Insert** as discussed above, and executes an inner loop until there is no improvement of the best solution for maxIterations successive iterations. The insertion step is iteratively applied within this loop until a local optimum is reached. Lastly, the shake step is applied. The pseudo code of CSCRatio algorithm is listed below (Algorithm 1). In order to reduce the search space (therefore, the execution time) of **CSCRatio_Insert**, in case that a non-included POI p is found infeasible to insert in any route, it is removed from the list of candidate POIs and added back, only after **Shake** has been applied.

Cluster Search Cluster Routes Algorithm. Given a route t of a TOPTW solution, any maximal sub-route in t comprising a sequence of nodes within the same cluster C is defined as a *Cluster Route (CR)* of t *associated with cluster* C and denoted as CR_C^t. The length of CR_C^t may be any number between 1 and $|C|$. Note that a route t of a TOPTW solution constructed by the ILS or CSCRatio algorithm may include more than one cluster route CR_C^t for the same cluster C, i.e., a tour t may visit and leave cluster C more than once. CSCRoutes

```
run the global k-means algorithm with k=numberOfClusters
construct the list listOfClusterSets
it1 ← $\frac{\text{maxIterations}}{4}$; it2 ← $\frac{2\cdot\text{maxIterations}}{4}$; it3 ← $\frac{3\cdot\text{maxIterations}}{4}$
while listOfClusterSets is not empty do
    remove all POIs visited in the currentSolution
    theClusterSetIdToInsert ← listOfClusterSets.pop
    RouteInitPhase(theClusterSetIdToInsert)
    startNumber ← 1; removeNumber ← 1; notImproved ← 0
    while notImproved < maxIterations do
        if notImproved < it2 then
            if notImproved < it1 then clusterParameter ← 1.3
            else clusterParameter ← 1.2
            end if
        else
            if notImproved < it3 then clusterParameter ← 1.1
            else clusterParameter ← 1.0
            end if
        end if
        while not local optimum do
            CSCRatio_Insert(clusterParameter)
        end while
        if currentSolution.profit > bestSolution.profit then
            bestSolution ← currentSolution ; removeNumber ← 1; notImproved ← 0
        else increase notImproved by 1
        end if
        if removeNumber > $\frac{\text{currentSolution.maxSize}}{2}$ then removeNumber ← 1
        end if
        Shake(removeNumber,startNumber)
        increase startNumber by removeNumber
        increase removeNumber by 1
        if startNumber ≥ currentSolution.sizeOfSmallestTour then
            decrease startNumber by currentSolution.sizeOfSmallestTour
        end if
    end while
end while
return bestSolution
```

Algorithm 1: **CSCRatio**(numberOfClusters,maxIterations)

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 t it cannot be revisited in the same route and therefore, for each cluster C there is only one cluster route in any route t associated with C. The only exception allowed is when the start and the end node of a route t belong to the same cluster C'. In this case, a route t may start and end with nodes of cluster C', i.e. C' may be visited twice in the route t and therefore, for a route t there might be two cluster routes $CR^t_{C'}$.

The insertion step **CSCRoutes_Insert** of CSCRoutes does not allow the insertion of a node p in a route t, if this insertion creates more than one cluster routes CR_C^t for some cluster C. Therefore, a POI cannot be inserted at any position in the route t. In the sequel, the description of insertion step **CSCRoutes_Insert** is given, based on the following assumptions. Consider w.l.o.g. that the start and end nodes in the TOTPW coincide (*depot*). If a route t contains two CRs associated with the cluster of the *depot*, then let CR_f^t be the first cluster route (starts at the *depot*) in t, and CR_l^t be the last cluster route (ends at the *depot*) in t. Also, assume that for each POI p ratio$_p$ is calculated as in ILS algorithm. Finally, consider for each route t, the list listOfClusters(t) containing any cluster C for which there is a nonempty CR_C^t. Given a candidate for insertion node p and a route t, **CSCRoutes_Insert** distinguishes among the following cases:

- cluster(p)=cluster(*depot*) and listOfClusters(t) contains only the cluster(*depot*). Then p can be inserted anywhere in the route, since the insertion would not violate the CR constraints.
- cluster(p)=cluster(*depot*) and listOfClusters(t) contains more than one cluster. Then p can be inserted anywhere in CR_f^t and in CR_l^t.
- cluster(p)$\neq$cluster(*depot*) and listOfClusters(t) contains only cluster(*depot*), then the insertion is feasible anywhere in t. If the insertion occurs, then a new CR will be created with p as its only POI.
- cluster(p)$\neq$cluster(*depot*) and listOfClusters(t) contains two or more clusters but not cluster(p). Then p can be inserted after the end of every CR in t. If the insertion occurs, then a new CR will be created with p as its only POI.
- cluster(p)$\neq$cluster(*depot*) and listOfClusters(t) contains two or more clusters and also includes cluster(p). Then p can be inserted anywhere in $CR_{\text{cluster}(p)}^t$.

The CSCRoutes algorithm is likely to create solutions of lower quality (w.r.t. overall profit), especially in instances featuring tight time windows. However, it significantly reduces the number of transfers among clusters and therefore it favors routes that include POIs of the same cluster. Thus, walking transfers are preferred while overly long travel distances are minimized. In effect, CSCRoutes is expected to perform better than ILS and CSCRatio with respect to execution time, since **CSCRoutes_Insert** is faster than **ILS_Insert** and **CSCRatio_Insert** (this is because the number of possible insertion positions for any candidate node is much lower).

4 Experimental Results

Test Instances. Montemanni and Gambardella [10] designed TOPTW instances based on previous OPTW instances of Solomon [11] and Cordeau et al. [1] (data sets for vehicle routing problems with time windows). Solomons instances comprise 100 nodes, with c1*, r1* and rc1* featuring much shorter time budget and tighter time windows than c2*, r2* and rc2* instances. Likewise, Cordeau et al. instances feature 48-288 nodes, constant time budget (=1000 min) and average time windows equal to 135 min and 269 min for pr01-10 and pr11-20 instances, respectively. All the aforementioned instances involve one, two, three and four tours.

The aforementioned instances, though, are not suitable for real-life TTDP problems, wherein: (a) POIs are typically associated with much wider, overlapping, multiple time windows; (b) POIs are densely located at certain areas, while isolated POIs are few; (c) visiting time at a POI is typically correlated with its profit value ; (d) the daily time budget available for sightseeing is typically in the order of a few hours per day (in contrast, most existing instances define unrealistically long time budgets). Along this line, we have created 100 new TOPTW instances (t*) with the following characteristics: the number of tours is 1-3; the number of nodes is 100-200; 80% of the nodes are located around 1-10 zones; the visiting time at any vertex is 1-120 min and proportional to the profit; regarding time windows, we assume that 50% of the nodes are open in 24h basis, while the remaining are closed either for one or two days per week (during their opening days, the latter are open 08:30-17:00); the daily time budget is set to 10h in t1* and 5h in t2* instances, respectively. The instances of Montemanni and Gambardella are available in
http://www.mech.kuleuven.be/en/cib/op/, while the t* instances in
http://www2.aegean.gr/dgavalas/public/op_instances/.

Results. All computations were carried out on a personal computer Intel Core i5 with 2.50 GHz processor and 4 GB RAM. Our tests compared our proposed algorithms against the best known real-time TOPTW approach (ILS). Clearly, mostly preferred solutions are those associated with high profit values, low number of transfers and reduced execution time. CSCRatio and CSCRoutes set the value of maxIterations equal to $\frac{400}{|\text{listOfClusterSets}|} \cdot \frac{k+1}{2 \cdot k}$. ListOfClusterSets is implemented by adding $\lceil \text{numberOfClusters}/k \rceil$ disjoint sets of k clusters which are randomly selected from the set of the clusters. The value of numberOfClusters is set to $N/10$.

Table 1 illustrates the average gaps among CSCRatio and ILS over all the existing and new test instances, with respect to profit, number of transfers and execution time; the existing instances have been tested on 1 to 4 tours. Positive gaps denote prevalence of our algorithm against ILS (the opposite is signified by negative gap values). CSCRatio yields significantly higher profit values, especially for instances with tight daily time budget and small number of tours (e.g. 0.79 in r1* and 2.04 in rc1*, for one tour). This is because ILS is commonly trapped in isolated areas with few high profit nodes, failing to explore remote areas with considerable numbers of fairly profited candidate nodes, As regards the number of transfers, CSCRatio clearly prevails, mainly when the time budget is prolonged (e.g. in c2*, r2* and rc2* instances), as it prioritizes the successive placement of nodes assigned to the same cluster into the tours. ILS and CSCRatio attain similar execution times in most cases, however the former clearly executes faster when examining instances with both long time budget and wide time windows. With regards to our new benchmark instances (i.e. t1* and t2*) CSCRatio achieves considerably higher profit gaps than ILS, especially when considering instances featuring tight time budgets (t2* instances). This improvement is attributed to the RouteInitPhase incorporated into both our proposed algorithms, which increases the probability of initially inserting high-profit nodes located on far-reached clusters (such nodes are typically overlooked by ILS itineraries due to the high travel time, hence, low insertion ratio). On the other hand, ILS performs better as regards the number of transfers yield on t2* instances (CSCRatio commonly explores areas far located from the depot, hence, it is forced to perform a number of inter-cluster transfers to connect those areas to the depot). Last, the two algorithms present comparable execution times.

Table 1. Average gaps between ILS and CSCRatio for Solomon, Cordeau et al., t* instances

	Profit Gap(%)				Transfers Gap (%)				Time Gap (%)			
Name	1	2	3	4	1	2	3	4	1	2	3	4
c1*	0.21	0.32	0.53	0.68	-0.2	-0.01	3.12	6.21	-40.8	8.45	35	24.9
c2*	0.84	0.79	0.29	0	19.1	12.6	12.3	20	-4.29	-18.2	-101	-398
r1*	0.79	0.91	-0.57	0.33	4.96	4.55	-0.11	2.71	-20.7	18.1	39.3	21.9
r2*	0.11	0.47	0.03	0	9.78	9.86	10.8	14	-4.88	-120	-305	-608
rc1*	2.04	0.87	0.81	-0.47	9.17	3.75	4.81	4.94	-1.07	36.3	34.7	44.7
rc2*	0.45	-0.34	0.32	0	5.49	1.83	0.48	5.49	11.9	-38.4	-197	-416
pr*	1.46	-0.02	0.4	0.9	-0.72	-9.99	4.5	4.62	29.1	27.4	7.44	-27.4
t1*	0.28				2.19				-5.27			
t2*	2				-13.2				8.33			

Table 2 illustrates the average gaps among CSCRoutes and ILS. The results indicate a trade-off between the profit and the number of transfers. In particular, ILS yields improved quality solutions as it inserts best candidate nodes freely, irrespective of their cluster assignment. This is especially true when considering instances which combine long time budgets with tight time windows (e.g. r2*), whereby CSCRoutes fails to use the time budget effectively, as it might get trapped within clusters, spending considerable amounts of time waiting for the nodes opening time, while not allowed to escape by visiting neighbor cluster nodes. This disadvantage is mitigated when the number of tours increases, as high-profit nodes are then more likely to be selected. On the other hand, CSCRoutes clearly improves on ILS with respect to the number of transfers due to its focal design objective to prohibit inter-cluster transfers. CSCRoutes also attains shorter execution times (excluding the c2*, r2* and rc2* instances for 4 tours), as it significantly reduces the search space on its insertion phase (i.e. in order to insert a new vertex between a pair of nodes that belongs to the same cluster, it only examines nodes assigned to the same cluster). As regards the new benchmark instances, ILS yields higher profit values than CSCRoutes in t1*, however, the performance gap is decreased compared to the results reported on previous instances. This is due to the wider and overlapping time windows chosen in t* instances, which diminishes the wait time (until opening) and allows more effective use of the budget time by CSCRoutes. CSCRoutes performs much better with respect to number of transfers and execution time. Interestingly, the results differ significantly on t2* instances, with CSCRoutes deriving solutions of considerably higher quality at the expense of increased number of transfers.

This is mainly due to some outlier values, which largely affect the average value. In those instances, CSCRoutes is initialized inserting a far-located high-profit vertex and is forced to traverse a number of intermediate clusters in order to connect it to the depot vertex. It is noted that CSCRoutes retains lead over ILS with regard to the execution time on t2* instances.

Table 2. Average gaps between ILS and CSCRoutes for Solomon, Cordeau et al., t* instances

	Profit Gap(%)				Transfers Gap (%)				Time Gap (%)			
Name	1	2	3	4	1	2	3	4	1	2	3	4
c1*	-1.65	-3.59	-1.03	-1.36	19.2	22.1	23.8	20.2	-21.1	29	38.2	31.2
c2*	-0.82	0.79	0.14	0	36	30.5	25.7	37.7	65.5	57.9	10.4	-170
r1*	-1.2	-1.27	-2.37	-2.15	23	21.7	16.6	22.2	0.1	36.5	49.4	38.6
r2*	-15.5	-10.3	-3.79	-1.21	56.3	55.4	52.6	46.4	76.7	25.1	-77.7	-284
rc1*	1.06	-1.8	-1.26	-1.86	16.7	11.9	14.8	14.4	10.7	50.3	42.9	52.9
rc2*	-9.5	-12.5	-8.21	-2.63	39.7	42.6	44.5	45.2	76	51.1	-40	-203
pr*	-8.11	-8.11	-5.44	-4.8	35.5	34.1	32.6	32.4	62.2	62.9	42.9	14.6
t1*	-0.52				5.31				22.2			
t2*	1.91				-4.5				4.59			

5 Conclusions

The comparison of CSCRatio over ILS demonstrated that CSCRatio achieves higher quality solutions in comparable execution time (especially when considering limited itinerary time budget), while also reducing the average number of transfers. As regards the CSCRoutes-ILS comparison, the former clearly prevails in situations where the reduction of inter-cluster transfers is of critical importance. The transfers gap, though, is achieved at the expense of slightly lower quality solutions. CSCRoutes achieves the best performance results with respect to execution time, compared to ILS and CSCRatio. Notably, the performance gap of our algorithms over ILS increases when tested on realistic TTDP instances, wherein nodes are located nearby each other and feature wide, overlapping time windows, while the daily time budget is 5-10h. We argue that our two cluster-based heuristics may be thought of as complementary TTDP algorithmic options. The choice among CSCRatio and CSCRoutes (when considering real-world online TTDP applications) should be determined by user-stated preferences. For instance, a user willing to partially trade the quality of derived solutions with itineraries more meaningful to most tourists (i.e. mostly walking between successive POI visits, rather than public transportation transfers) should opt for the CSCRoutes algorithm.

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

References

1. J-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119, 1997.
2. L.M. Gambardella, R. Montemanni, and D. Weyland. Coupling ant colony systems with strong local searches. *European Journal of Operational Research*, 220(3):831 – 843, 2012.
3. B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.
4. N. Labadi, R. Mansini, J. Melechovský, and R. Wolfler Calvo. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1):15 – 27, 2012.
5. N. Labadi, J. Melechovský, and R. Wolfler Calvo. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics*, 17:729–753, 2011.
6. G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193 – 207, 1990.
7. Z. Li and X. Hu. The team orienteering problem with capacity constraint and time window. *The Tenth International Symposium on Operations Research and Its Applications (ISORA 2011)*, pages 157–163, August 2011.
8. A. Likas, N. Vlassis, and J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451 – 461, 2003.
9. S.-W. Lin and V. F. Yu. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 217(1):94 – 107, 2012.
10. R. Montemanni and L. M. Gambardella. An ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 34(4):287–306, 2009.
11. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35:254–265, 1987.
12. F. Tricoire, M. Romauch, K. F. Doerner, and R. F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351 – 367, 2010.
13. P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011.
14. P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Comput. Oper. Res.*, 36:3281–3290, December 2009.
15. P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: An or opportunity. *Operational Research Insight*, 20(3):21–27, 2007.