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

FP7 - Information and Communication Technologies

Grant Agreement No: 288094
Collaborative Project
Project start: 1 November 2011, Duration: 36 months

D3.5.2 - Context-aware multi-modal daily routes for tourists and their empirical assessment

Workpackage: WP3 - Algorithms for Multimodal Human Mobility
Due date of deliverable: 30 June 2013
Actual submission date: 07 July 2013
Responsible Partner: CTI
Contributing Partners: CTI, ETHZ, KIT, TomTom

Nature: ☒ Report ☐ Prototype ☐ Demonstrator ☐ Other

Dissemination Level:
☒ PU: Public
☐ PP: Restricted to other programme participants (including the Commission Services)
☐ RE: Restricted to a group specified by the consortium (including the Commission Services)
☐ CO: Confidential, only for members of the consortium (including the Commission Services)

Keyword List: Tourist Trip Design Problem, Route planning, Orienteering Problem, Time Window, Time Dependent Team Orienteering, Time dependent travel time, Multi-modal, Public transport.

The eCOMPASS project (www.ecompass-project.eu) is funded by the European Commission, DG CONNECT (Communications Networks, Content and Technology Directorate General), Unit H5 - Smart Cities & Sustainability, under the FP7 Programme.

The eCOMPASS Consortium

Computer Technology Institute & Press 'Diophantus' (CTI) (coordinator), Greece

Centre for Research and Technology Hellas (CERTH), Greece

Eidgenössische Technische Hochschule Zürich (ETHZ), Switzerland

Karlsruher Institut fuer Technologie (KIT), Germany

TOMTOM INTERNATIONAL BV (TOMTOM), Netherlands

PTV PLANUNG TRANSPORT VERKEHR AG. (PTV), Germany

Document history			
Version	Date	Status	Modifications made by
0.1	7.06.2013	Table of Contents draft	Damianos Gavalas, CTI
0.2	21.06.2013	First Draft	Julian Dibbert, KIT
0.3	28.06.2013	Second Draft	Damianos Gavalas, CTI
1.0	29.06.2013	Sent to internal reviewers	Damianos Gavalas, CTI
1.1	04.07.2013	Reviewers' comments received	Damianos Gavalas, CTI
1.2	05.07.2013	Reviewers' comments incorporated (sent to PQB)	Damianos Gavalas, CTI
1.3	06.07.2013	PQB's comments received	Damianos Gavalas, CTI
1.4	07.07.2013	Final (sent to the Project Officer)	Christos Zaroliagis, CTI

Deliverable manager

- Damianos Gavalas, CTI

List of Contributors

- Julian Dibbelt, KIT
- Damianos Gavalas, CTI
- Felix König, TomTom
- Vlasios Kasapakis, CTI
- Charalambos Konstantopoulos, CTI
- Konstantinos Mastakas, CTI
- Grammati Pantziou, CTI
- Andreas Paraskevopoulos, CTI
- Nikolaos Rousias, CTI
- Nikolaos Vathis, CTI
- Christos Zaroliagis, CTI

List of Evaluators

- Dionisis Kehagias, CERTH
- Sandro Montanari, ETHZ

Summary

The main aim of this deliverable is development of models and algorithmic solutions for context-aware multi-modal daily route planning problems for tourists visiting multiple points of interests (POIs) optimized for mobile devices. Those route planning problems, collectively known as tourist trip design problems (TTDP), involve deriving personalized recommendations for daily sightseeing itineraries for tourists visiting any urban destination. We firstly tackle a simplistic variant of TTDP considering constant travel times among POIs (i.e. exclusively walking transfers). Building upon that, we then take into account time-dependent (i.e. multimodal) travel times in our TTDP modeling. Our prototyped algorithms have been evaluated and tested upon both existing and new test instances. We have also used validation scenarios comprising real POI sets compiled from the Athens (Greece) area and calculated multimodal travel times based on the metropolitan transit network of Athens.

Contents

1 Introduction **5**
1.1 Objectives and scope of D3.5.2 . . . 5
1.2 The Tourist Trip Design Problem . . . 5
1.3 Structure of the Deliverable . . . 7

2 Related work **8**

3 Modeling the Tourist Trip Design Problem as a Team Orienteering Problem with Time Windows **11**
3.1 Introduction . . . 11
3.2 Mathematical formulation . . . 11
3.3 The ILS algorithm [50] . . . 12
3.4 Cluster based TOPTW heuristics . . . 15
3.4.1 Cluster Search Cluster Ratio Algorithm . . . 16
3.4.2 Cluster Search Cluster Routes Algorithm . . . 18
3.5 Experimental Results . . . 20
3.5.1 Test instances . . . 20
3.5.2 Results . . . 21
3.6 Conclusions . . . 33

4 Modeling the Tourist Trip Design Problem as a Time-Dependent Team Orienteering Problem with Time Windows **34**
4.1 Introduction . . . 34
4.2 TDTOPTW heuristics . . . 35
4.2.1 Time dependent insertion feasibility . . . 37
4.2.2 The Time Dependent CSCRoutes (TDCSCRoutes) algorithm . . . 39
4.2.3 The SlackCSCRoutes algorithm . . . 40
4.2.4 The Average Travel Times CSCRoutes (AvgCSCRoutes) algorithm . . . 43
4.3 Solving the Generalized Tourist Trip Design Problem . . . 44
4.4 Preprocessing Multimodal Travel Time Distances . . . 45
4.4.1 Multimodal Profile Queries . . . 46
4.4.2 Acceleration . . . 48
4.4.3 Results . . . 48
4.4.4 Outlook . . . 48
4.5 Experimental Results . . . 49
4.5.1 Test Instances . . . 49
4.5.2 Results . . . 50
4.6 Conclusions and Future Work . . . 58

Appendices **63**

Appendix A Analytical Results for TOPTW Algorithms **63**

1 Introduction

This deliverable presents the research results obtained by the project partners in the first 20 months of the project with respect to tourist route planning in urban areas. It describes the models and algorithms developed so far for the problems relevant to WP3. It also demonstrates experimental results aiming to assess the quality of the proposed solutions.

1.1 Objectives and scope of D3.5.2

The aim of WP3 is to provide novel methods for route planning in urban public transportation networks, considering the environmental impact as a main optimization objective. The present deliverable is the outcome of Task 3.5 "Algorithms for context-aware multi-modal daily routes for tourists & their empirical assessment". This task aims at developing models and algorithmic solutions for context-aware multi-modal daily route planning problems for tourists visiting multiple points of interests (POIs) optimized for mobile devices. Those route planning models tailored to tourists are collectively known as tourist trip design problems (TTDP). In the context of Task 3.5, we aim at developing models and algorithms outperforming the state-of-the-art techniques in terms of computational time and quality of derived solutions.

1.2 The Tourist Trip Design Problem

A TTDP [52] refers to a route-planning problem for tourists interested in visiting multiple POIs. TTDP solvers derive daily tourist tours i.e., ordered visits to POIs, which respect tourists' constraints and POIs' attributes. The main objective of the problem discussed 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 days/hours, entrance fees, weather conditions) 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 set of candidate POIs, each associated with the following attributes: (i) a location (i.e. geographical coordinates), (ii) time windows (TW) (i.e. opening hours for each day of the week), (iii) 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 (iv) a visiting time (i.e. the anticipated duration of visit of a user at the POI ,derived from the average anticipated duration and the user's potential interest for that particular POI).
- The travel time among POIs; these may either be constant (i.e. considering exclusively walking transfers) or time-dependent (i.e. considering multimodal tranfers).
- The number k of routes that must be generated, based upon the period of stay (number of days) of the user at the tourist destination.
- 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 (see Figure 1). Therefore, a TTDP solution should feature POI recommendations that match tourist preferences and near-optimal feasible route scheduling. The team orienteering problem with time windows (TOPTW), introduced by P. Vansteenwegen [48], may serve as a basic model to formulate

TTDP, when considering constant travel times among locations. In the TOPTW a set of locations is given, each associated with a profit, a visiting time and a time window. Each location may be visited only once, while the aim is to maximize the overall profit collected by a fixed number of routes. The length of each route (time allowed for sightseeing within a single day) must not exceed a given time budget.

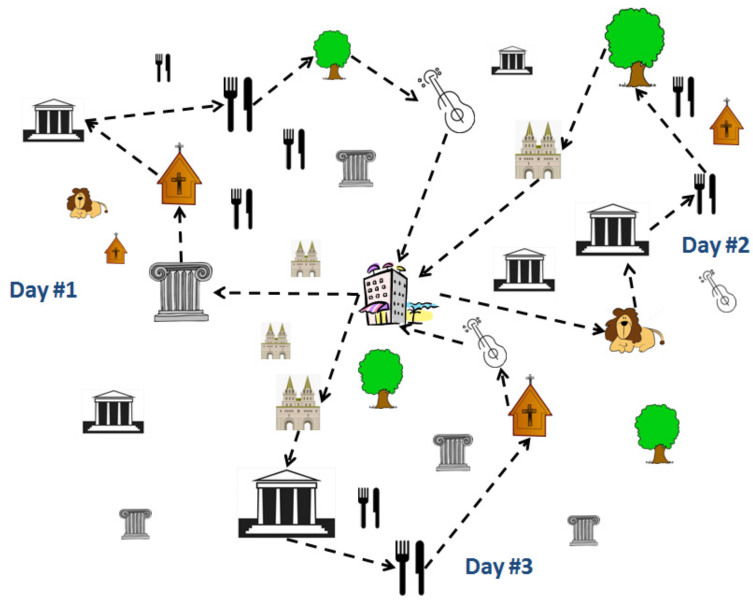


Figure 1: Illustration of TTDP

In addition to the above-listed TTDP input data, in the context of eCOMPASS, TTDP's modeling is approached also considering (see Figure 2):

- a number of user-defined preferences and restrictions (e.g. tourist interests, period of visit, accommodation location, daily time allowance for sightseeing, etc)
- multi-modal routing among POIs, i.e. tourists are assumed to use all modes of transport available at the tourist destination, including public transportation, walking and bicycle.
- end users are assumed to be either web or mobile users; in addition to location, several contextual parameters may be taken into account in recommending sub-optimal itineraries to mobile users (e.g. day/time, weather conditions, traffic conditions, etc).

Clearly, TOPTW modeling captures several aspects of TTDP. Nevertheless, it overlooks the time-dependency (i.e. multimodality) of urban transfers among tourist sites. The Time Dependent TOPTW (TDTOPTW) is the problem that best matches TTDP requirements among all problems found in the relevant algorithmic research literature, as it allows modeling transfers via urban transit networks (in addition to walking and other available transportation modes).

In this deliverable we firstly tackle a simplistic variant of TTDP considering constant travel times among POIs (i.e. exclusively walking transfers). In this context, we introduce two efficient TOPTW heuristic algorithms deriving high-quality solutions, as evidenced from prototype testing upon both existing and new test instances. Building upon those algorithms, we then proceed a step further taking into account multimodal travel times in our TTDP modeling and introducing three novel TDTOPTW algorithms. The latter have been validated on benchmarks comprising real POI sets compiled from Athens (Greece) along with calculated multimodal travel times based on the timetable of the metropolitan transit network of Athens.

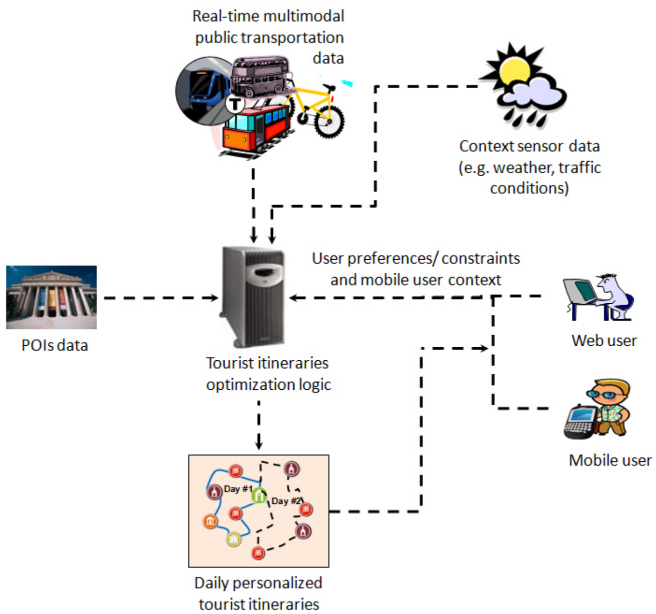


Figure 2: User groups, input data and recommended itineraries in TTDP

1.3 Structure of the Deliverable

The remainder of this document is organized as follows: Section 2 overviews algorithmic approaches relevant to the TTDP. Section 3 introduces a model for TTDP based on TOPTW, analyzing and evaluating two novel heuristic algorithmic approaches. Section 4 incorporates time dependency on urban transfers, while discussing and testing three novel heuristic algorithms for TDTOPTW.

2 Related work

The algorithmic and operational research literature features many route planning problem modeling approaches, effectively simplified versions of the TTDP. One of the simplest problems that may serve as a basic model for TTDP is the orienteering problem (OP) [49]. The OP is based on the orienteering game, in which 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 OP clearly relates to the TTDP: the OP locations are POIs associated with a score (i.e. user satisfaction) and the goal is to maximize the score collected within a given time budget (time allowed for sightseeing in a single day).

Extensions of the OP have been successfully applied to model the TTDP. The team orienteering problem (TOP) extends the OP considering multiple routes (i.e. daily tourist itineraries). The (T)OP with time windows (TOPTW) considers visits to locations within a predefined time window (this allows modeling opening and closing hours of POIs). Several further generalizations exist that allow even more detailed modeling of the TTDP, e.g. taking into account multiple user constraints (MCTOPTW) such as the overall budget that may be spent for POI entrance fees.

Among the above, TOPTW is the most well-studied problem with respect to the TTDP as it closely matches most of TTDPs modeling requirements. TOPTW is a special case of the Vehicle Routing Problem with Profits (VRPP) and time windows [2, 22]. In VRPP visiting the whole set of nodes is not compulsory; a profit is collected when visiting a node, while collecting the profits is distributed over several vehicles with limited capacity.

TOPTW is NP-hard and APX-hard (e.g. see [26], [32], [22]). Exact solutions for TOPTW are feasible for instances with very restricted number of locations (e.g. see the work by Z. Li and X. Hu [36], which is used on networks of up to 30 nodes). Given the complexity of the problem, the main body of TOPTW literature involves mainly heuristic algorithms based on simulated annealing [38], local search [50] and ant colony system (ACS) [40].

Labadi et al. [29] 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 (not included in a path) 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. [38] 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 ILS heuristic proposed by Vansteenwegen et al. [50] is the fastest known algorithm proposed for TOPTW [49]. ILS defines an “insertion” and a “shake” step. The insertion step adds, one by one, new visits to a route, ensuring that all subsequent visits (those scheduled after the insertion place) remain feasible, i.e. they still satisfy their time window constraint. For each visit i that can be inserted, the cheapest insertion cost is determined. For each of these visits the heuristic calculates a ratio, which represents a measure of how profitable is to visit i versus the time delay this visit incurs. Among them, the heuristic selects the one with the highest insertion ratio. The shake step is used to escape from local optima. During this step, one or more visits are removed in each route in search of non-included visits that may either decrease the route time length or increase the overall collected profit.

Montemanni and Gambardella proposed an ACS algorithm [40] 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. The algorithm comprises two phases:

- Construction phase: Ants are sent out sequentially; when at node i, an ant chooses probabilistically the next node j to visit (i.e. to include into the route) based on two factors:
  - The pheromone trail τ_{ij} (i.e. a measure on how good it has been in the past to include arc (i, j) in the solution).
  - The desirability n_{ij}, (a node j is more desirable when it is associated with high profit, it is not far from i, and its time window is used in a suitable way).
- Local search: performed upon the solutions derived from construction phase, aiming at taking them down to a local optimum.

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 [18] a modified ACS framework (Enhanced ACS) is presented and implemented for the TOPTW to improve the results of ACS.

Labadi et al. [30], [31] recently proposed a method that combines the greedy randomized adaptive search procedure (GRASP) with the evolutionary local search (ELS). GRASP generates independent solutions (using some randomized heuristic) further improved by a local search procedure. ELS generates multiple copies of a starting solution (instead of a single copy generated in ILS) using a random mutation (perturbation) and then applies a local search on each copy to yield an improved solution. GRASP-ELS 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 [31].

Tricoire et al. [47] 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 motivation behind this modelling is to facilitate individual route planning of field workers and sales representatives. 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. Several further generalizations exist that allow even more detailed modeling of the TTDP, e.g. taking into account multiple user constraints (MCTOPTW) such as the overall budget that may be spent for POI entrance fees.

Vansteenwegen et al. [49] argue that a detailed comparison of TOPTW solution approaches (i.e. ILS, ACS and the algorithm of Tricoire et al. [47]), 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, while ACS and the approach of Tricoire et al. [47] (2010) have the advantage of obtaining higher quality solutions.

Notably, TOPTW does not consider time-dependency in calculating cost of edges, i.e. travelling times among vertices. Time dependency is useful for modeling transfers among nodes through multimodal public transportation, hence it is considered particularly relevant to the TTDP. Time-dependent graphs have been used in almost all variants of the orienteering problems, from the basic OP to the TOPTW.

Time Dependent OP (TDOP) was introduced by Formin and Lingas [17]. TDOP is MAX-SNP-hard since a special case of TDOP, time-depenent maximum scheduling problem is MAX-SNP-hard [46]. An exact algorithm for solving TDOP is given by Li et al [35] using a mixed integer programming model and a pre-node optimal labeling algorithm based on the idea of dynamic programming. Moreover, Li [34] proposes an exact algorithm for TDTOP based on dynamic programming principles. However, both algorithms are of exponential complexity, hence, not appropriate for real-time applications, especially when considering complex transportation networks and relatively large POI datasets. Fomin and Lingas [17] give a $(2+\epsilon)$ approximation algorithm for rooted and unrooted TDOP ("which runs in polynomial time if the ratio R between the maximum and minimum travelling time between any two sites is constant"). When considering unrooted TDOP its running time is $O((2R^2(\frac{2+\epsilon}{\epsilon}))!\frac{2R^2}{\epsilon}n^{2R^2(\frac{2+\epsilon}{\epsilon})+1})$, and for rooted TDOP its running time increases by the multiplicative factor $O(\frac{Rn}{\epsilon})$. (The key idea is derived from Spieksma's algorithm [46] for Job Interval Selection Problem). They use a divide-and-conquer approach. First the problem is split in smaller ones. Exact solutions are found to each smaller problem and later combined (stitch) to obtain an approximate solution.

Abbaspour et al. [1] investigated a variant of Time Dependent OP with Time Windows (TDOPTW) in urban areas, where the nodes are partitioned into the POIs (associated with profits and time windows) and multimodal transportation stops which do not have profit. A genetic algorithm is proposed for the problem that uses as a subroutine another genetic algorithm for solving the shortest path problem between POIs.

The Time-Dependent TOP with Time Windows (TDTOPTW) is the problem that best matches TTDP requirements among all problems and approaches surveyed in this section. TDTOPTW is particularly complex as it adds time dependency of arcs to TOPTW. Zenker and Ludwig [53] described a tourism-inspired problem that refers to TDTOPTW and presented ROSE, a mobile application assisting pedestrians to locate events and locations, moving through public transport connections. ROSE incorporates three main services: recommendation, route generation and navigation. The authors identified the route planning problem to solve and they described it as a multi-constrained destination recommendation with time windows using public transportation. However, no algorithmic solution to this problem has been proposed.

The work of Garcia et al. [20, 21] is the first to address algorithmically the TDTOPTW extending previous work on TDOPTW [19]. 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 (employing a time-dependent Dijkstra algorithm), 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 introduces four additional variables per vertex and is based on a fast evaluation of the possible insertion of an extra POI. The authors argue that their algorithm is suitable for real-time applications, requiring slightly longer computational time than fast TOPTW algorithms (i.e. ILS) to derive sufficiently quality solutions. However, the solutions yield using their second approach are not perturbed so as to reduce computational overhead; hence, those solutions are sensitive to the quality of the insertion phase. Even more so, the algorithm's modeling is 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) wherein arrival/departure times on intermediate stops is subject to dynamic traffic fluctuations and several other non-predictable incidents. Herein, we propose an algorithmic approach that alleviates this assumption and is applicable to realistic urban transit networks. For a full overview of the relevant literature, the reader is referred to Section 4 of D3.1 "Multimodal tourist trip design problems").

3 Modeling the Tourist Trip Design Problem as a Team Orienteering Problem with Time Windows

3.1 Introduction

The TTDP is typically dealt with online web and mobile applications with strict execution time restrictions [45, 51]. Hence, only highly efficient heuristic approaches are eligible for TTDP solvers. The most efficient known heuristic is based on Iterated Local Search (ILS) [50], 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. These issues are discussed in more detail in Section 3.3.

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 (plane) areas featuring high density of 'good' candidate vertices (these 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 vertices, which typically require public transportation rides (these transfers are costly and usually less attractive to tourists than short walking transfers).

The remainder of this section is organized as follows: Section 3.2 provides the mathematical formulation of TOPTW, while Section 3.3 presents the ILS approach in more detail. Section 3.4 presents our novel cluster-based heuristics, while Section 3.5 discusses the experimental results compiled from executing ILS as well as our algorithms on several test instances. Finally, Section 3.6 concludes this section.

3.2 Mathematical formulation

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 $B_i, i = 1, \ldots, k$. 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 ($[\text{open}_{im}, \text{close}_{im}], m = 1, 2, \ldots, k$) (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 $r_1, r_2, \ldots, r_k$ starting from 1 and ending at N, each with overall duration limited by the time budget $B_i, i = 1, \ldots, k$ (i.e. r_i has length at most $B_i, i = 1, \ldots, k$) , that maximize the overall profit collected by visited POIs in all routes.

Then TOPTW can be formulated as an integer programming problem as follows [50]:

$$\max \sum_{m=1}^{k} \sum_{i=2}^{N-1} \text{profit}_i y_{im}, \tag{1}$$

s.t.

$$\sum_{m=1}^{k} \sum_{j=2}^{N} x_{1jm} = \sum_{m=1}^{k} \sum_{i=1}^{N-1} x_{iNm} = k, \tag{2}$$

$$\sum_{i=1}^{N-1} x_{irm} = \sum_{j=2}^{N} x_{rjm} = y_{rm}, \text{ for all } r = 2, \ldots, N-1, m = 1, \ldots, k \tag{3}$$

$$\sum_{m=1}^{k} y_{rm} \leq 1, \text{ for all } r = 2, \ldots, N-1, \tag{4}$$

$$\sum_{i=1}^{N-1} (\text{visit}_i y_{im} + \sum_{j=2}^{N} \text{travel}_{ij} x_{ijm}) \leq B_m \text{ for all } m = 1, \ldots, k, \tag{5}$$

$$\text{start}_{im} + \text{visit}_i + \text{travel}_{ij} - \text{start}_{jm} \leq C(1 - x_{ijm}) \text{ for all } i, j = 1, \ldots, N, m = 1, \ldots, k \tag{6}$$

$$\text{open}_{im} \leq \text{start}_{im} \text{ for all } i = 1, \ldots, N, m = 1, \ldots, k \tag{7}$$

$$\text{start}_{im} \leq \text{close}_{im}, i = 1, \ldots, N, m = 1, \ldots, k \tag{8}$$

$$x_{ijm}, y_{im} \in \{0, 1\}, \text{ for all } i, j = 1, \ldots, N, \ m = 1, \ldots, k \tag{9}$$

where x_{ijm} is equal to 1 if, in route m, i is followed by j, or equal to 0 otherwise, y_{im} is equal to 1 if i is visited in route m or equal to 0 otherwise; start_{im} is the start of the visit at node i in route m and C is a large number (larger enough from the parameters of the problem).

The objective function (1) is to maximize the total profit of visited POIs. Constraint (2) ensures that each of the k routes starts at node 1 and ends at node N. Constraint (3) ensures that each route is connected. Constraint (4) guarantees that each node belongs to at most one route. Constraint (5) indicates that the time budget is not violated. Constraint (6) ensures that the sequence of starting times of visits at the nodes inside a route is feasible. Constraints (7, 8) indicate that the start of a visit can only take place during the time window.

3.3 The ILS algorithm [50]

The ILS heuristic proposed by Vansteenwegen et al. [50] is the fastest known algorithm proposed for TOPTW [49]. ILS 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 start_{it} $+\text{visit}_i + \text{travel}_{ip} \leq \textit{close}_{pt}$ 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). 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 visits 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.

The ILS algorithm ([50]) initializes the parameters startNumber and removeNumber of the shake step to 1 and loops up to a specified number of times (150) as long as the profit of the best solution is not improved. Inside the loop, the insertion step is applied until a local optimum is reached. If the current solution's profit is larger than the profit of the best solution, the current solution is kept as the best solution and parameter removeNumber is reset to one. In the sequel, the shake step is applied. After the application of the shake step, the values of its parameters are adapted as follows: the value of startNumber is increased by the value of removeNumber and the value of removeNumber is increased by one. If startNumber is greater than or equal to the size of the smallest route in the current solution, then startNumber is decreased by this size. If removeNumber equals to $\frac{N}{3k}$ then it is reset to one. The pseudo code of the ILS algorithm is listed below (Algorithm 1).

Algorithm 1 ILS (Vansteenwegen et al. [50])

```
maxIterations ← 150
maxNumberToRemove ← N/3k
startNumber ← 1; removeNumber ← 1; notImproved ← 0
while notImproved < maxIterations do
    while not local optimum do
        ILS_Insert
    end while
    if currentSolution.profit > bestSolution.profit then
        bestSolution ← currentSolution
        removeNumber ← 1
        notImproved ← 0
    else increase notImproved by 1
    end if
    Shake(removeNumber,startNumber)
    increase startNumber by removeNumber
    increase removeNumber by 1
    if startNumber ≥ currentSolution.sizeOfSmallestRoute then
        decrease startNumber by currentSolution.sizeOfSmallestRoute
    end if
    if removeNumber == maxNumberToRemove then
        removeNumber ← 1
    end if
end while
return bestSolution
```

The execution of the algorithm is demonstrated in Figure 3. Figure 3(a) illustrates the insertion phase (for one route) when considering two non-included candidate nodes (k and l) for insertion. Shift_k is minimized when k's insertion between i and j is considered (Figure 3(b),(c),(d) illustrate insertion options for k). Similarly, shift_l is minimized when l's insertion between j and n is considered (Figure 3(e)). The candidate node finally selected is l, as $\text{ratio}_l > \text{ratio}_k$, namely node l is found relatively more profitable to visit for the time cost it adds to the overall route time. During the shake step, visit i is removed (Figure 3(f)) and replaced in the next insertion step by the -originally not included- visit k (Figure 3(g)), found to have larger ratio value.

To the best of our knowledge, ILS is the fastest known algorithm for solving the TOPTW

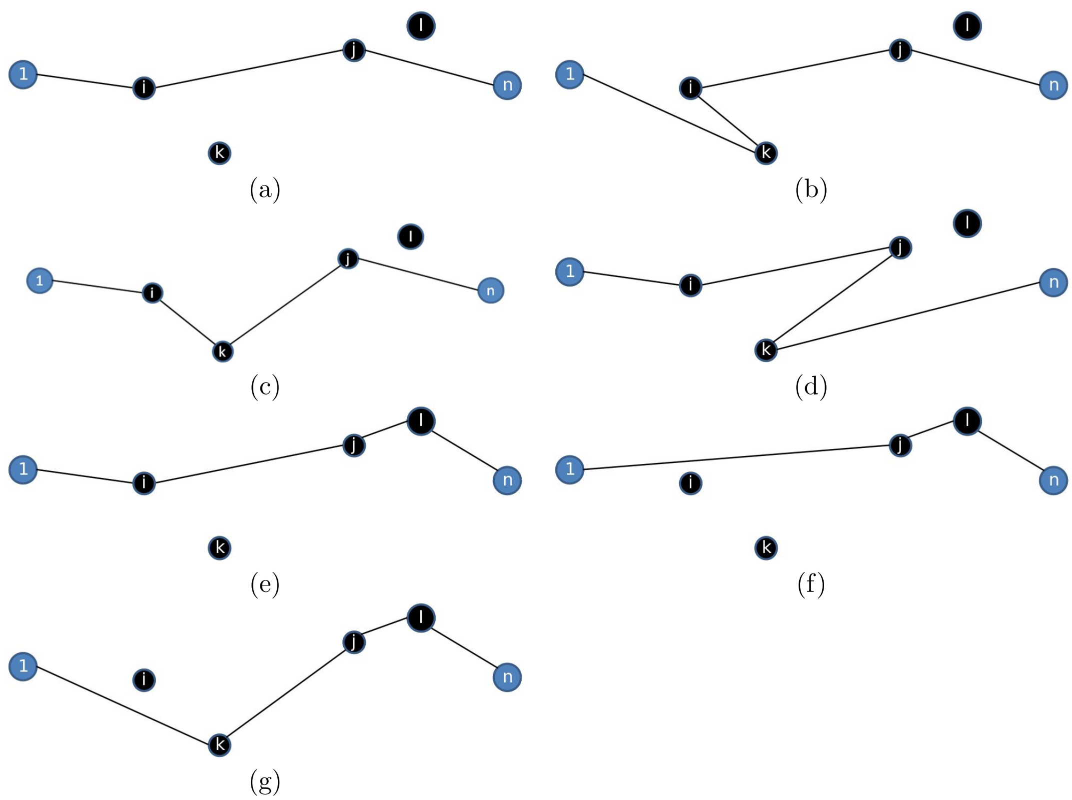
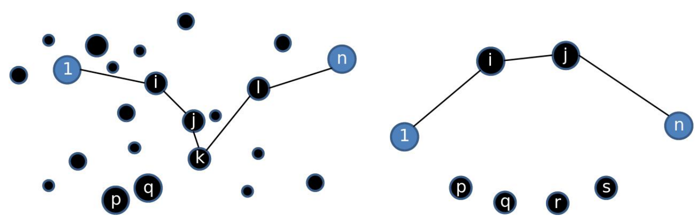


Figure 3: (a-e) ILS insertion phase; (f) ILS shake phase, (g) insertion of k. Circle sizes denote nodes' score.

offering a fair compromise in terms of speed versus deriving routes of reasonable quality. However, it presents the following weaknesses:

- 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. For instance, in Figure 4(a), ILS inserts i, l, j and k. Although p and q have larger score value, they are not selected on the first four insertion steps since they are associated with large shift values. On the next step, p and q can no longer be inserted, since the insertion would violate the route feasibility constraint.
- 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). For instance, in Figure 4(b), the itinerary $\{1, p, q, r, s, n\}$ would yield more profit and fully utilize the available time budget, compared to the chosen solution $\{1, i, j, n\}$.



(a) (b)

Figure 4: Weakness of ILS

3.4 Cluster based TOPTW heuristics

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 certain cluster, our algorithms encourage visits to others 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; such transfers may add 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 [37, 3] to organize the set of POIs into an appropriate (based on the network topology) number of clusters (numberOfClusters). Global k-means is an effective global clustering approach, which minimizes the clustering error and employs the k-means algorithm as a local search procedure. The algorithm obtains an optimal solution for the clustering problem through applying a series of local searches using the k-means algorithm. Once the clusters of POIs have been formed during an offline 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 (Figure 5). 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 as described in the following subsections.

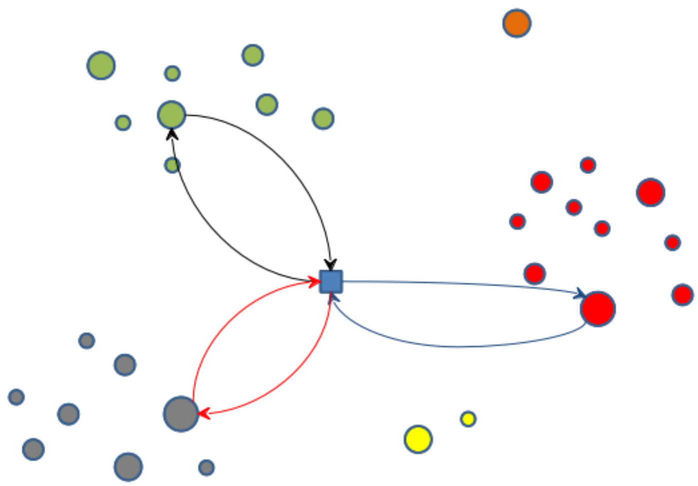


Figure 5: Illustration of the RouteInitPhase

3.4.1 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

$$\text{shiftCluster}_p = \frac{\text{shift}_p}{\text{clusterParameter}}$$

in the case that cluster(p) coincides with cluster(i) or cluster(j), where 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**. In this way, 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 in the current solution and not $\frac{N}{3k}$ as in ILS. 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 can be increased without increasing the overall algorithm's execution time. Therefore, CSCRatio may exercise a larger maxIterations value.

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 2).

Algorithm 2 CSCRatio(numberOfClusters,maxIterations)

```
run the global k-means algorithm with k=numberOfClusters
construct the list listOfClusterSets
it1 ← maxIterations/4; it2 ← 2·maxIterations/4; it3 ← 3·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 > currentSolution.sizeOfLargestTour/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
```

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. Figure 6 illustrates an example solution obtained by CSCRatio.

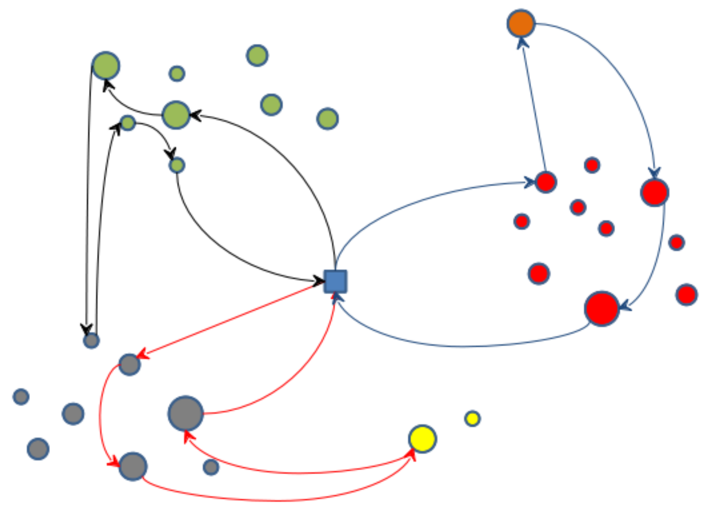


Figure 6: Example of a CSCRatio solution

3.4.2 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 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 $CR_{C'}^t$.

The insertion step **CSCRoutes_Insert** of the CSCRoutes algorithm 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 CR 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)≠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)≠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)≠cluster($depot$) and listOfClusters(t) contains two or more clusters and also includes cluster(p). Then p can be inserted anywhere in $CR^t_{\text{cluster}(p)}$.

The pseudo code of **CSCRoutes_Insert** (Algorithm 3) follows.

Algorithm 3 CSCRoutes_Insert

```
for each candidate POI p do
    clusterID←cluster(p)
    for each route t do
        if clusterID==cluster(depot) then
            if listOfClusters(t) contains only cluster(depot) then
                Search all possible insert positions in t for the least shift_p
            else
                Search all possible insert positions in CR^t_f and CR^t_l for the least shift_p
            end if
        else // clusterID≠cluster(depot)
            if listOfClusters(t) contains only cluster(depot) then
                Search all possible insert positions in t for the least shift_p
            else
                if listOfClusters(t) doesn't contain clusterID then
                    Search all possible positions in t that are the end of a CR, for the least shift_p
                else Search all possible insert positions in CR^t_clusterID for the least shift_p
                end if
            end if
        end if
    end for
end for
Insert the POI q with the highest ratio.
Update times, maxShifts and listOfClusters.
```

Note that similarly to the CSCRatio algorithm when a non-included POI p is infeasible to insert in any route, then p is removed from the list of candidates and re-examined, only after **Shake** has been applied.

Like CSCRatio algorithm, CSCRoutes executes a loop for a number of times equal to the size of the listOfClusterSets. Within the loop, firstly, all POIs in the current solution's routes are removed and the route initialization phase is executed. Secondly, the algorithm initializes the parameters startNumber and removeNumber of **Shake** to 1 and executes an inner loop up to a specific number of times (maxIterations) while the profit of the best solution is not improved. Within this loop, the insertion step **CSCRoutes_Insert** is applied until a local optimum is reached. At the end, the shake step is applied. The pseudo code of CSCRoutes Algorithm 4 is given below.

The CSCRoutes algorithm is likely to create solutions of lower quality than ILS (i.e. decreased 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. In this way, walking transfers are preferred while overly long travel distances are minimized. At the same time, the 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). Figure 7 illustrates an example solution obtained by CSCRoutes.

Algorithm 4 CSCRoutes(numberOfClusters,maxIterations)

```
run the global k-means algorithm with k=numberOfClusters
construct the list listOfClusterSets
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
        while not local optimum do
            CSCRoutes_Insert
        end while
        if currentSolution.profit > bestSolution.profif then
            bestSolution ← currentSolution ; removeNumber ← 1; notImproved ← 0
        else increase notImproved by 1
        end if
        if removeNumber > currentSolution.sizeOfLargestTour/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
```

3.5 Experimental Results

3.5.1 Test instances

Montemanni and Gambardella [40] designed TOPTW instances based on previous OPTW instances of Solomon [44] (data sets for vehicle routing problems with time windows: c10*, r10* and rc10*) and Cordeau et al. [7] (10 multi-depot vehicle routing problems: pr1pr10). They also added 27 extra instances based on Solomon (c20*, r20* and rc20*) and 10 instances based on Cordeau et al. (pr11pr20). Cordeau et al. instances have up to 288 customers and much wider time windows than in Solomon's problems. All the aforementioned instances involve one, two, three and four tours. Optimal solutions are available for some of those test instances. Herein, we compare the performance of our heuristics against the best-known algorithm suitable to real-time TTDP applications, i.e. ILS [50].

The aforementioned instances allow a fair comparison of our proposed heuristics against published results, yet, they do not represent suitable examples of real-life TTDP problems. In such problems (a) POIs are typically associated with much wider, overlapping, multiple time windows (e.g. Mon closed, Tue-Fri 08:30-16:00, Sat-Sun 09:00-18:00); (b) POIs' locations are statistically dependent, i.e. typical tourist destination topologies feature dense concentration of POIs at certain areas, while isolated POIs are rare; (c) visiting time at a POI is typically correlated with its profit value (e.g. POIs associated with high profit value are expected to take long to visit); (d) the time available for sightseeing (daily time budget) is typically in the order of a few hours per day (in contrast, Cordeau et al. and Solomon r2*/rc2* instances allow time budget up to 16.5 hours, while Solomon c2* instances up to 56.5 hours, which is certainly unrealistic).

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 vertices is 100-200, which is considered a fair estimation of available POIs on medium-to-large scale urban tourist destinations; 80% of the vertices are located around 1-10 virtual 'centers' (the distances of vertices from their randomly assigned center follow a Gaussian distribution); a 20% of the vertices is set at a random location on the plane; the profit associated with vertices is 1-100, while visiting time at any vertex is 1-120 min

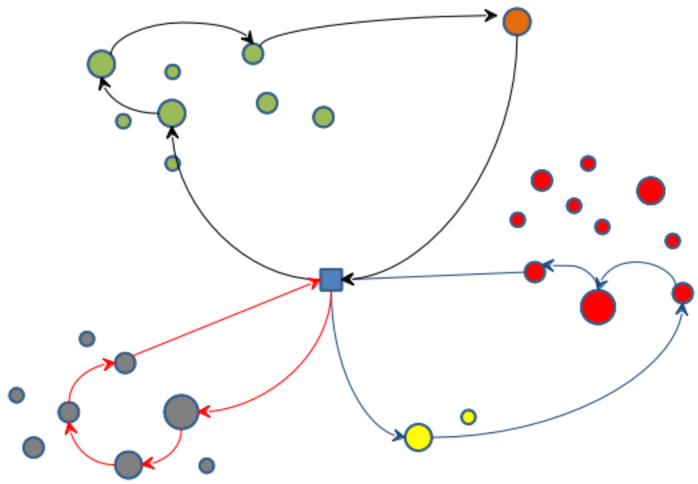


Figure 7: Example of a CSCRoutes solution

(visiting time is proportional to the profit); regarding time windows, we assume that 50% of the vertices are open in 24h basis (e.g. squares, parks and landmarks non open to visitors), while the remaining are closed either on weekends (15%) or one day per week, either Monday (15%), Tuesday (10%) or Wednesday (10%) (during their opening days, the non-24h vertices are open 08:30-17:00); the daily time budget is set to 10h (510-1210 min) in t1* and 5h (840-1140 min) in t2* instances.

Table 1: TOPTW Instances

<table>
<tr><th>Reference</th><th>Based on</th><th># of instances</th><th>N</th><th>B</th><th>Average TW</th><th>k</th></tr>
<tr><td rowspan="4">Montemanni et al. [40]</td><td>Solomon (c1*, r1* and rc1*)</td><td>29</td><td>100</td><td>c1*:1236
r1*:230
rc1*:240</td><td>c1*:321
r1*:87
rc1*:85</td><td>1,2,3,4</td></tr>
<tr><td>Cordeau et al.(pr01-10)</td><td>10</td><td>48-288</td><td>1000</td><td>135</td><td>1,2,3,4</td></tr>
<tr><td>Solomon (c2*, r2* and rc2*)</td><td>27</td><td>100</td><td>c2*:3390
r2*:1000
rc2*:960</td><td>c2*:921
r2*:454
rc2*:370</td><td>1,2,3,4</td></tr>
<tr><td>Cordeau et al.(pr11-20)</td><td>10</td><td>48-288</td><td>1000</td><td>269</td><td>1,2,3,4</td></tr>
<tr><td rowspan="2">Gavalas et al. [23]</td><td>t1*</td><td>50</td><td>100-200</td><td>600</td><td>1000</td><td>1,2,3</td></tr>
<tr><td>t2*</td><td>50</td><td>100-200</td><td>300</td><td>997</td><td>1,2,3</td></tr>
</table>

Table 1 overviews the available TOPTW test instances. For every set of instances, the corresponding reference is given, along with the name of the original instances the set is based on. The number of instances, vertices (N) and tours (k) as well as the daily time budget (B) are also presented.

The benchmark instances of Montemanni and Gambardella are available in `http://www.mech.kuleuven.be/en/cib/op/`, while the t* instances are available in `http://www2.aegean.gr/dgavalas/public/op_instances/`.

3.5.2 Results

All computations were carried out on a personal computer Intel Core i5 with 2.50 GHz processor and 4 GB RAM. Our tests aim at comparing our proposed algorithms against the best known real-time TOPTW approach (ILS), which yields high quality solutions, while being suitable for real-time TTDP applications. Reported results compare ILS against CSCRatio and CSCRoutes with respect to the following aspects: (a) overall collected profit, (b) number of transfers, namely links among far-located vertices (those are associated with pairs of vertices, each assigned to separate cluster, based on our offline clustering procedure), and (c) execution (CPU) time required to derive a solution. In addition to our proposed algorithms, ILS has been also implemented to reliably measure the number of transfers of its respective solutions and ensure fair comparison with respect to execution (CPU) time required to derive solutions; the overall collected profit values corresponding to ILS are those published in [50]. Clearly, mostly preferred solutions are those associated with high profit values (higher profit values denote higher quality solutions), low number of transfers (transfers typically involve use of multi-modal transport, hence, they are considered less attractive for tourists, as they are more expensive and less eco-friendly options than walking) and reduced execution time (as this denotes improved suitability for real-time TTDP applications). All three algorithms have been coded in C++. 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 $\left\lceil \frac{\text{numberOfClusters}}{k} \right\rceil$ disjoint sets of k clusters which are randomly selected from the set of the clusters.

In Appendix A, we provide the analytical results and compare ILS, CSCRatio and CSCRoutes based on the benchmark instances of Solomon (Tables 23, 25, 27 and 29) and Cordeau et al. (Tables 24, 26, 28 and 30). We provide results for one (Tables 23 and 24), two (Tables 25 and 26), three (Tables 27 and 28) and four tours (Tables 29 and 30) over the same sets of instances. The first two columns show the instance's name and the number of clusters derived from our global k-means clustering algorithm (the latter is proportional to the instances' number of vertices, i.e. $\frac{N}{10}$). The next three sets of column triads correspond to the results yield for ILS, CSCRatio and CSCRoutes, respectively. Total collected profit, number of inter-cluster transfers and execution time are reported for each algorithm.

The comparison results between ILS and CSCRatio for Solomon and Cordeau et al. instances are summarized in Tables 2 and 3, respectively, for different number of tours. Positive gaps denote predominance of our algorithm against ILS. The opposite (i.e. prevalence of ILS solution) is signified by negative gap values. CSCRatio yields significantly higher profit values, especially for instances with tight B and small number of tours (e.g. 0.79 in r1* and 2.04 in rc1*, for one tour, in Table 2). 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 vertices, due to prohibitively large travelling time and the limited time budget (see relevant discussion in Section 3.3). The null (0) values mostly appearing in c2*, r2* and rc2* instances for 3 or 4 tours indicate that both approaches derive the optimum solution since k and B are large enough to accommodate all vertices into the solution. It should be noted that average profit gaps higher than one (1) unit are considered as significant improvement, since ILS achieves average gap less than 1.8% from the best known solution on these instances [50]. It is noted that the best known solution (in many cases known to be the optimal solution) is calculated by the ant colony system of Montemanni and Gambardella [40]; however, this algorithm requires prohibitively long time to derive solutions (increased by a factor of more than 100 compared to ILS [49]), which makes it inappropriate for online TTDP applications. As regards the number of transfers, CSCRatio clearly prevails, mainly when B is prolonged (e.g. in c2*, r2* and rc2* instances), as it prioritizes the successive placement of vertices assigned to the same cluster into the tours. On the other hand, ILS disregards the geographic position of successive vertices, as long as they maximize the insertion ratio. ILS and CSCRatio attain similar execution times in most cases, however the former clearly executes faster when examining instances with both large B and k values (in those cases, it appears that the CSCRatio shake step commonly yields slightly better solutions, thereby reinitializing new iteration rounds in search of improved solutions and prolonging the execution time).

The comparison results between ILS and CSCRoutes for Solomon and Cordeau et al. instances are summarized in Tables 4 and 5, respectively. The results indicate a trade-off between profit and number of transfers. In particular, ILS yields better results with respect to profit as it inserts best candidate nodes freely, irrespective of their cluster assignment. This is especially true when considering instances which combine large B 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 vertices' opening time, while not allowed to 'escape' by visiting neighbour cluster vertices. This disadvantage is mitigated when k increases, as high-profit vertices are then more likely to be selected. On the other hand, CSCRoutes clearly surpasses 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 k=4), as it significantly reduces the search space in its insertion phase (i.e. in order to insert a new vertex between a pair of vertices that belongs to the same cluster, it only examines vertices assigned to the same cluster).

The comparison results between CSCRatio and CSCRoutes for Solomon and Cordeau et al. instances are summarized in Tables 6 and 7, respectively, where negative values indicate prominence of CSCRatio. As expected, CSCRatio obtains better results in terms of profit as it enables broader exploration of the search space on its insertion phase. On the other hand, CSCRatio performs worse with respect to the number of transfers (as it follows a more relaxed approach when considering connecting vertices that belong to different clusters) and execution time (since it involves a broader search space). The execution time gap increases in favor of CSCRoutes on instances with large B values (e.g. c2*, r2* and rc2*) as their respective solutions accommodate higher numbers of vertices, hence, the insertion iterations (which are much more time consuming in CSCRatio) increase accordingly.

The analytical results obtained by ILS, CSCRatio and CSCRoutes on our new benchmark instances (i.e. t1* and t2*) are illustrated in Tables 31 and 32, respectively (see Appendix A). Table 8 compares ILS against CSCRatio. The latter achieves considerably higher profit gaps compared to the previously examined instances, especially when considering instances featuring tight time budgets (t2* instances). This agrees with the results compiled for c1*, r1* and rc1* Solomon instances, which possess similar characteristics. This improvement is attributed to the RouteInitPhase incorporated into both our proposed algorithms, which increases the likelihood of initially inserting high-profit vertices located on far-reached clusters (such vertices are typically overlooked by ILS itineraries due to the high travel time, hence low 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 9 compares ILS against CSCRoutes. ILS yields higher profit values in t1* instances, 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 (for instance, in t243 instance ILS collects far less profit but performs only 2 transfers as opposed to 4 transfers performed by CSCRoutes), 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.

Last, Table 10 compares CSCRatio against CSCRoutes (negative values indicate prominence of CSCRatio). The general picture is that CSCRatio prevails with respect to solutions' quality, while CSCRoutes yields improved results with regard to both the number of transfers involved and the execution time measured.

Table 2: Comparison between ILS and CSCRatio for Solomon instances

	1 tour Gap(%)			**2 tours Gap(%)**			**3 tours Gap(%)**			**4 tours Gap(%)**		
Name	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time
c101	-3.13	-14.29	9.62	0	20	29.86	1.27	-10	55.97	1	14.81	40.19
c102	0	0	-34.74	0	-27.27	37.8	0	5	36.9	1.83	20.83	33.77
c103	0	0	-99.01	1.43	7.14	-33.44	0	15	18.91	0.87	10.71	-3.06
c104	5	22.22	-69.53	0	0	5.24	0	-6.67	-2.19	-1.64	-15.79	45.61
c105	0	-22.22	-25.77	0	0	24.31	1.19	23.08	35.28	2.91	11.54	57.69
c106	0	-12.5	-28.71	0	0	6.74	1.19	0	41.49	1.92	-4	21.35
c107	0	0	-36.54	0	0	14.8	0	0	59.08	0.91	13.04	-18.76
c108	0	0	-27.42	0	0	-4.01	1.11	-4.55	72.59	0	-4.35	38.27
c109	0	25	-54.95	1.41	0	-5.29	0	6.25	-2.97	-1.69	9.09	8.77
Average	0.21	-0.2	-40.78	0.32	-0.01	8.45	0.53	3.12	35.01	0.68	6.21	24.87
c201	2.38	12.5	33.09	1.43	12.5	24.84	0.57	22.86	-14.03	0	-12.5	-342.77
c202	0	14.29	39.87	0	20.69	-33.33	1.71	26.32	-21.54	0	24.56	-375.5
c203	0	26.32	10.88	0.7	10.71	-60.97	-0.57	20.45	-113.53	0	21.57	-380.2
c204	1.05	35.29	-53.55	-0.68	7.14	-38.06	0	30.23	-200.76	0	28.57	-434.13
c205	0	7.69	-17.24	0	0	38.69	0	7.69	-109.12	0	21.57	-386.87
c206	1.1	20	-22.43	2.08	11.76	-41.28	1.69	0	-183.09	0	32.73	-464.2
c207	2.2	29.41	8.51	1.38	27.27	-9.83	-1.1	-9.09	-67.2	0	26.42	-396.21
c208	0	7.14	-33.48	1.37	10.53	-25.47	0	0	-94.9	0	16.67	-408.04
Average	0.84	19.08	-4.29	0.79	12.58	-18.18	0.29	12.31	-100.52	0	19.95	-398.49
r101	0.55	-16.67	-4.92	3.94	16.67	31.11	-1.25	5.88	46.82	-0.5	5.26	71.46
r102	0	0	-9.91	-1.38	-9.09	12.07	-2.19	-14.29	23.28	-0.12	9.09	34.62
r103	1.75	14.29	-64.36	0.19	7.69	-11.99	0	6.67	37.59	1.14	0	-3.71
r104	1.35	33.33	-42.74	0.74	0	-14.74	0.26	-15.38	45.43	1.06	30.43	28.16
r105	0	0	32.59	2.79	0	32.14	-2.13	15.79	64.62	2.45	8.33	0.74
r106	0	0	-18.52	-0.95	0	37.71	-2.09	0	32.15	1.26	0	-14.09
r107	2.08	28.57	-62	-0.95	9.09	6.63	-0.54	0	33.33	-1.62	9.52	-3.54
r108	3.7	0	-10	1.28	18.18	-11.88	0.51	0	58.48	-1.53	-5.88	8.5
r109	0	0	9.68	1.61	-9.09	31.38	-0.29	0	55.41	0.92	-22.22	19.86
r110	0	0	7.64	-1.36	11.11	39.34	0.98	0	22.48	1.03	9.09	22.23
r111	0	0	-24.03	0.56	0	46.12	0	0	36.03	-1.18	0	56.83
r112	0	0	-62.28	4.47	10	19.12	-0.13	0	15.54	1.06	-11.11	42.13
Average	0.79	4.96	-20.74	0.91	4.55	18.08	-0.57	-0.11	39.26	0.33	2.71	21.93
r201	-0.25	17.86	30.75	-1.54	14.81	-55.01	0.28	12.31	-72.45	0	6.67	-278.91
r202	1.25	22.22	32.23	2.52	6.38	-34.55	0	1.69	-128.43	0	10.77	-384.12
r203	0.31	0	45.26	-0.36	13.04	-79.61	0	14.29	-279.19	0	6.67	-476.06
r204	-1.49	30.43	-51.82	-0.14	15.79	-157.17	0	28.57	-450.16	0	29.82	-765.66
r205	-2.79	0	43.87	-0.37	2.5	-52.25	0	4.76	-237.99	0	8.7	-514.29
r206	0	-4.76	-30.53	0.36	6.82	-150.67	0	-3.85	-355.99	0	3.23	-606.39
r207	2.02	4.76	-39.54	0.42	8.33	-167.69	0	7.94	-393.71	0	19.4	-755.38
r208	1.31	0	5.71	0	16.28	-265.57	0	20.41	-441.42	0	36.73	-1066.17
r209	-0.65	-4.55	-36.4	1.86	16.67	-82.42	0	1.89	-301.95	0	16.95	-690.19
r210	1.25	12.5	13.78	1.32	2.38	-57.16	0	8.93	-319.31	0	6.06	-502.22
r211	0.2	29.17	-67.03	1.13	5.41	-216.01	0	21.43	-378.73	0	8.93	-650
Average	0.11	9.78	-4.88	0.47	9.86	-119.83	0.03	10.76	-305.39	0	13.99	-608.13
rc101	0	0	16.13	0	0	70.36	1.66	25	60.14	-1.89	18.75	68.31
rc102	0	0	16.24	-1.21	-14.29	31.27	-0.43	15.38	66.95	-0.34	31.58	55.43
rc103	-0.75	33.33	-15	-0.58	-14.29	4.73	2.14	-9.09	34.21	1.9	13.33	20.66
rc104	1.35	20	-60.24	1.59	11.11	32.64	-0.73	0	25.47	0.49	-14.29	50.34
rc105	10.41	0	19.83	4.14	0	44.65	-0.92	0	37.78	-1.78	6.25	32.7
rc106	4.6	20	23.39	5.02	27.27	35.53	0.74	0	28.96	-0.92	12.5	20.02
rc107	0.73	0	11.29	-0.19	11.11	47.13	0.94	-7.14	18.26	-0.21	0	48.56
rc108	0	0	-20.19	-1.83	9.09	23.71	3.04	14.29	5.52	-1	-28.57	61.56
Average	2.04	9.17	-1.07	0.87	3.75	36.25	0.81	4.81	34.66	-0.47	4.94	44.7
rc201	-0.38	0	34.13	2.91	9.76	-3.01	1.54	0	-86.41	0	-3.17	-265.81
rc202	4.76	26.32	28.74	-1.78	-8.82	4.17	-0.12	15.52	-130.42	0	14.93	-335.38
rc203	-0.42	-38.46	1.45	-0.7	8.57	-53.43	0	-2.38	-214.86	0	22.22	-457.84
rc204	-0.81	21.43	-29.11	0.6	0	-120.95	0	4.76	-338.29	0	18.37	-658.21
rc205	0.6	11.76	22.87	-1.3	5.56	10.92	0.06	9.26	-143.26	0	5.08	-328.85
rc206	1.05	5.26	-4.99	-1.2	0	-57.07	0.76	-9.3	-208.54	0	-10.17	-417.29
rc207	-0.11	11.76	13.95	-1.5	-6.9	-73.88	0.35	-17.95	-138.03	0	-7.02	-417.67
rc208	-1.06	5.88	27.9	0.25	6.45	-13.68	0	3.92	-312.53	0	3.64	-447.81
Average	0.45	5.49	11.87	-0.34	1.83	-38.37	0.32	0.48	-196.54	0	5.49	-416.11
Total Av.	0.7	7.77	-11.03	0.53	5.5	-21.2	0.17	5.11	-83.43	0.11	8.67	-220.74

Table 3: Comparison between ILS and CSCRatio for Cordeau et al. instances

	1 tour Gap(%)			**2 tours Gap(%)**			**3 tours Gap(%)**			**4 tours Gap(%)**		
Name	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time
pr01	0	0	29.36	4.67	-18.18	-24.22	-2.68	40	-111.9	1.55	0	-312.5
pr02	1.04	-60	43.13	1.36	-50	40.23	-1.89	9.52	11.76	0.59	13.04	-61.46
pr03	2.34	-22.22	13.12	-0.14	-23.53	11.22	-1.16	10.34	31.5	1.64	3.7	13.51
pr04	3.8	50	46.19	-3.59	0	45.24	0.5	-3.23	66.99	0.07	7.5	52.46
pr05	-4.17	14.29	35.3	3.56	28.13	28.13	1.92	-7.69	27.89	0.3	12.5	69.42
pr06	2.97	-23.08	28.13	-3.31	0	50.56	-1.67	20.93	68.55	0.88	4	39.94
pr07	0	16.67	-10	0.54	-42.86	28.67	1.12	-23.08	48.22	-0.71	-5.26	-32.11
pr08	-3.67	28.57	48.94	-1.63	10.53	36.74	-0.65	-3.7	27.92	0.39	8.11	-0.04
pr09	1.52	20	27.45	-5.88	-18.18	53.4	5.16	6.9	54.44	3.22	-6.98	34.49
pr10	-2.04	-7.69	24.45	5.38	0	44.56	-1.83	-7.89	57.65	0.17	10.71	23.16
pr11	3.03	-60	28.68	-3.87	0	-40.65	0.47	15.38	-185.53	0	0	-310.53
pr12	0.7	0	17.46	0	-60	3.9	2.77	15.79	-69.58	1.25	8.7	-126.41
pr13	-0.67	-14.29	10.5	5.55	-21.43	33.7	-0.1	-10	12.7	-0.63	-3.03	35.27
pr14	4.77	33.33	8.28	1.95	22.73	30.3	3.59	-10.34	28.39	3.6	-8.82	29.97
pr15	-0.31	-5.88	23.51	-2.22	3.45	49.42	-0.74	5.56	20.69	-0.72	21.15	15.28
pr16	3.22	-10	69.25	-3.06	7.69	37.99	1.56	19.44	31.56	-1.01	-2.5	28.02
pr17	0.87	22.22	21.46	-0.64	9.09	57.43	-1.86	35	-61.05	-0.9	18.18	-112.71
pr18	9.19	-20	51.16	1.71	-61.54	-9.29	-2.75	-9.09	14.2	2.37	-3.57	14.58
pr19	2.2	11.11	34.75	-5.86	14.29	35.75	7.51	-7.41	17.18	4.87	12.82	16.61
pr20	4.39	12.5	31.45	5.11	0	34.26	-1.25	-6.45	57.24	1.14	2.13	34.64
Average	1.46	-0.72	29.13	-0.02	-9.99	27.37	0.4	4.5	7.44	0.9	4.62	-27.42

Table 4: Comparison between ILS and CSCRoutes for Solomon instances

	1 tour Gap(%)			**2 tours Gap(%)**			**3 tours Gap(%)**			**4 tours Gap(%)**		
Name	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time
c101	-6.25	28.57	11.54	-6.78	46.67	49.28	-1.27	25	56.25	-4	44.44	32.31
c102	0	0	-16.84	0	0	46.34	-1.12	25	18.52	-0.92	29.17	42.28
c103	-2.56	0	-57.43	0	14.29	-1.64	-2.08	20	30.83	-0.87	35.71	16.05
c104	2.5	22.22	-60.16	-1.33	14.29	23.6	0.99	6.67	20.83	0.82	10.53	43.64
c105	-2.94	22.22	-2.06	-6.25	25	31.94	-1.19	38.46	41.3	-0.97	23.08	68.78
c106	-2.94	25	-22.77	-3.23	29.41	14.54	-1.19	30.43	42.28	-0.96	12	18.67
c107	0	25	-8.65	-7.46	15.38	39.47	-2.22	21.05	68.78	-1.82	17.39	1.27
c108	-2.7	25	-4.03	-4.48	43.75	34.11	-2.22	40.91	74.29	-1.82	-4.35	33.16
c109	0	25	-29.73	-2.82	10	23.54	1.05	6.25	-9.35	-1.69	13.64	24.64
Average	-1.65	19.22	-21.13	-3.59	22.09	29.02	-1.03	23.75	38.19	-1.36	20.18	31.2
c201	0	37.5	70.89	2.14	37.5	66.37	0	37.14	47.45	0	20	-147.8
c202	-2.2	35.71	81.28	0.7	37.93	55.9	0	28.95	42.13	0	43.86	-145.03
c203	-4.26	47.37	76.72	0.7	46.43	52.95	-0.57	47.73	0.17	0	43.14	-164.69
c204	2.11	41.18	57.19	0	46.43	56.85	0.56	48.84	-24.38	0	46.43	-180.89
c205	-1.11	23.08	58.87	-0.69	5.56	72.81	1.13	19.23	-12.42	0	35.29	-170.71
c206	-1.1	33.33	54.65	2.08	17.65	51.25	1.69	4.76	-10.59	0	41.82	-206.61
c207	1.1	41.18	67.42	1.38	36.36	53.25	-1.66	9.09	26.74	0	37.74	-179.31
c208	-1.08	28.57	56.59	0	15.79	54.16	0	9.52	14.38	0	33.33	-168.53
Average	-0.82	35.99	65.45	0.79	30.46	57.94	0.14	25.66	10.44	0	37.7	-170.45
r101	-1.1	16.67	-14.75	-1.52	41.67	40.56	-6.86	35.29	48.09	-4.16	31.58	69.16
r102	-1.4	16.67	6.31	-1.38	18.18	40	-2.77	7.14	37.8	-1.61	31.82	44.35
r103	1.05	14.29	-25.74	-1.75	38.46	11.64	-1.67	20	58.43	-2.28	36.36	30.05
r104	2.02	16.67	-29.06	-1.86	0	17.05	-2.48	15.38	57.12	-1.7	26.09	37.6
r105	-3.64	40	37.04	-1.86	30	39.29	-2.79	42.11	65.22	-3.13	33.33	38.14
r106	-4.78	16.67	-7.41	-4.54	16.67	51.58	-2.78	0	23.74	-2.87	15	15.09
r107	0.35	14.29	-19	-1.13	9.09	24.1	-0.4	7.14	41.85	-2.16	23.81	22.89
r108	2.02	0	14.12	0.55	18.18	10.72	-2.66	7.14	67.95	-2.34	0	24.46
r109	-6.16	57.14	23.39	-3.61	36.36	53.99	-3.15	25	70.72	-1.39	11.11	26.57
r110	0	0	24.31	-1.75	22.22	43.96	-0.56	18.75	29.25	-0.34	31.82	29.77
r111	0.68	33.33	7.75	0.56	0	64.13	-0.79	14.29	50.48	-3.21	20	70.77
r112	-3.39	50	-15.79	3.11	30	40.34	-1.58	7.14	41.54	-0.64	5.56	54.05
Average	-1.2	22.98	0.1	-1.27	21.74	36.45	-2.37	16.62	49.35	-2.15	22.21	38.58
r201	-39.59	64.29	83.78	-29.81	64.81	67.9	-16.9	55.38	39.03	-8.16	52	-139.7
r202	-10.45	62.96	83.32	-12.2	57.45	53.19	-6.86	54.24	2.91	-4.53	43.08	-127.94
r203	-6.73	54.55	87.03	-9.73	56.52	22.87	-4.87	52.38	-78.93	-0.14	46.67	-209.51
r204	-2.33	56.52	67.99	-5.28	50	-12.74	-0.96	57.14	-146.3	0	47.37	-319.7
r205	-30.83	64.29	90.08	-16.67	50	54.76	-3.7	53.97	-60.78	-0.48	53.62	-227.41
r206	-14.76	52.38	70.61	-7.64	54.55	31.33	-2.06	48.08	-132.93	0	41.94	-405.02
r207	-11.66	52.38	67.17	-3.5	58.33	-28.91	-0.34	57.14	-105.66	0	49.25	-331.18
r208	-0.75	41.18	80.51	-2.67	53.49	-30.92	0	51.02	-135.92	0	42.86	-558.65
r209	-22.57	54.55	68.7	-11.75	60.42	54.02	-3.36	47.17	-66.85	0	42.37	-306.54
r210	-15.14	58.33	82.33	-6.15	52.38	49.18	-2.61	50	-76.08	0	51.52	-178.15
r211	-15.44	58.33	62.36	-7.45	51.35	15.85	0	51.79	-92.7	0	39.29	-325.27
Average	-15.48	56.34	76.72	-10.26	55.39	25.14	-3.79	52.57	-77.66	-1.21	46.36	-284.46
rc101	0	0	15.05	-1.87	-14.29	74.31	1.66	25	63.51	-2.27	31.25	60.02
rc102	2.7	20	30.77	0.61	0	58.59	-0.86	7.69	70.55	-2.95	26.32	69.49
rc103	0.38	33.33	5	0	-14.29	30.18	-2.41	0	52.76	-1.69	13.33	47.83
rc104	1.35	20	-32.53	-1.77	22.22	42.41	-2.19	9.09	23.87	-0.49	7.14	59.38
rc105	9.05	0	25.86	-5.23	25	48.62	-3.06	18.18	44.35	-3.21	12.5	43.32
rc106	4.6	20	30.65	1.31	27.27	55.84	-1.18	15.38	34.79	-2.52	18.75	38.79
rc107	-4.74	20	17.74	-5.44	22.22	47.75	-2.82	21.43	49.28	0.53	20	45.6
rc108	-4.86	20	-6.73	-2.01	27.27	44.33	0.79	21.43	4.27	-2.3	-14.29	58.44
Average	1.06	16.67	10.73	-1.8	11.93	50.25	-1.26	14.78	42.92	-1.86	14.38	52.86
rc201	-17.18	47.37	79.25	-20.77	53.66	64.05	-15.45	49.06	24.09	-7.95	46.03	-96.49
rc202	-2.04	47.37	78.35	-18.96	47.06	65.69	-13.11	55.17	-7.38	-4.87	49.25	-197.95
rc203	-6.15	23.08	77.27	-13.29	45.71	53.19	-6.5	40.48	-99.32	-0.29	47.62	-318.82
rc204	0.36	28.57	67.25	-2.96	26.92	28.29	-1.33	38.1	-81.14	0	32.65	-312.44
rc205	-18.45	41.18	75.53	-24.33	52.78	68.4	-17.06	48.15	8.32	-7.6	47.46	-97.8
rc206	-12.67	47.37	74.12	-11.3	41.18	26.22	-7.32	39.53	-25.62	0	47.46	-184.07
rc207	-13.5	41.18	77.12	-6.79	34.48	44.67	-4.9	38.46	-11.15	-0.35	47.37	-242
rc208	-6.36	41.18	79.28	-1.43	38.71	58.32	0	47.06	-127.86	0	43.64	-171.04
Average	-9.5	39.66	76.02	-12.48	42.56	51.1	-8.21	44.5	-40.01	-2.63	45.19	-202.58
Total Av.	-4.88	32.27	33.44	-4.79	31.22	40.17	-2.75	29.84	3.37	-1.56	31	-88.33

Table 5: Comparison between ILS and CSCRoutes for Cordeau et al. instances

	1 tour Gap(%)			**2 tours Gap(%)**			**3 tours Gap(%)**			**4 tours Gap(%)**		
Name	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time
pr01	-22.37	50	59.63	-7.22	45.45	34.38	-2.68	55	-114.29	0.78	41.18	-181.94
pr02	-2.86	20	69.81	-4.39	25	69.25	-8.68	42.86	49.06	-6.61	30.43	-2.35
pr03	-9.11	33.33	53.06	-11.06	35.29	64.99	-6.24	41.38	50.56	-6.45	22.22	57.31
pr04	-4.92	61.11	73.15	-7.65	20	73.01	-4.1	22.58	74.99	-7.51	32.5	60.25
pr05	-22.57	50	71.09	-9.2	46.88	71.55	-3.54	46.15	59.72	-4.62	30.36	81.3
pr06	-12.27	30.77	57.81	-12.84	29.17	72.77	-11.34	34.88	81.05	-8.96	36	47.46
pr07	0	16.67	34.17	0	-14.29	67.54	-2.81	23.08	70.49	-4.76	36.84	-17.2
pr08	-14.25	28.57	73.33	-9.3	47.37	69.1	-11.28	25.93	67.15	-5.37	37.84	55.52
pr09	-4.12	50	55.77	-13.61	40.91	71.57	-6.73	34.48	78.62	-2.67	30.23	59.31
pr10	-8.72	46.15	60	-7.17	30.77	69.16	-10.93	18.42	76.99	-9.43	30.36	58.24
pr11	-2.73	40	36.76	-2.58	40	12.9	-2.37	30.77	-73.68	0	23.53	-266.67
pr12	-5.34	40	60.28	-5.64	40	42.75	-0.33	47.37	33.6	1.44	30.43	-6.7
pr13	-6.44	14.29	50.21	-0.92	42.86	65.54	-5.93	25	55.44	-8.39	36.36	64.39
pr14	-3.53	11.11	58.34	-6.27	22.73	63.12	1.09	24.14	57.72	-4.38	26.47	65.48
pr15	-6.9	47.06	73.29	-12.88	51.72	76.36	-4.17	41.67	54.74	-8.8	53.85	49.81
pr16	-6.08	40	85.91	-16.31	46.15	70.75	-8.32	36.11	55.66	-7.89	35	63.86
pr17	-4.34	55.56	62.35	-4.81	36.36	72.83	-3.34	40	1.74	-1.8	40.91	-38.83
pr18	-14.82	40	77.1	-9.92	15.38	57.47	-9.87	22.73	53.61	-2.51	17.86	45.15
pr19	-2.4	11.11	62.51	-10.47	42.86	68.54	0.24	22.22	51.83	-4.68	30.77	46.52
pr20	-8.42	25	69.14	-10.04	37.5	65.28	-7.4	16.13	72.49	-3.36	25.53	50.09
Average	-8.11	35.54	62.19	-8.11	34.11	62.94	-5.44	32.55	42.87	-4.8	32.43	14.55

Table 6: Comparison between CSCRatio and CSCRoutes for Solomon instances

	1 tour Gap(%)			2 tours Gap(%)			3 tours Gap(%)			4 tours Gap(%)		
Name	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time
c101	-3.23	37.5	2.13	-6.78	33.33	27.69	-2.5	31.82	0.65	-4.95	34.78	-13.16
c102	0	0	13.28	0	21.43	13.73	-1.12	21.05	-29.12	-2.7	10.53	12.84
c103	-2.56	0	20.9	-1.41	7.69	23.83	-2.08	5.88	14.7	-1.72	28	18.55
c104	-2.38	0	5.53	-1.33	14.29	19.37	0.99	12.5	22.53	2.5	22.73	-3.61
c105	-2.94	36.36	18.85	-6.25	25	10.09	-2.35	20	9.3	-3.77	13.04	26.21
c106	-2.94	33.33	4.62	-3.23	29.41	8.37	-2.35	30.43	1.34	-2.83	15.38	-3.41
c107	0	25	20.42	-7.46	15.38	28.96	-2.22	21.05	23.71	-2.7	5	16.86
c108	-2.7	25	18.35	-4.48	43.75	36.66	-3.3	43.48	6.23	-1.82	0	-8.29
c109	0	0	16.28	-4.17	10	27.39	1.05	0	-6.2	0	5	17.4
Average	-1.86	17.47	13.37	-3.9	22.25	21.79	-1.54	20.69	4.79	-2	14.94	7.04
c201	-2.33	28.57	56.49	0.7	28.57	55.25	-0.57	18.52	53.91	0	28.89	44.03
c202	-2.2	25	68.87	0.7	21.74	66.92	-1.69	3.57	52.39	0	25.58	48.47
c203	-4.26	28.57	73.88	0	40	70.77	0	34.29	53.25	0	27.5	44.88
c204	1.04	9.09	72.12	0.69	42.31	68.75	0.56	26.67	58.64	0	25	47.41
c205	-1.11	16.67	64.92	-0.69	5.56	55.65	1.13	12.5	46.24	0	17.5	44.4
c206	-2.17	16.67	62.96	0	6.67	65.49	0	4.76	60.93	0	13.51	45.66
c207	-1.08	16.67	64.39	0	12.5	57.43	-0.56	16.67	56.19	0	15.38	43.71
c208	-1.08	23.08	67.48	-1.35	5.88	63.46	0	9.52	56.07	0	20	47.14
Average	-1.65	20.54	66.39	0.01	20.4	62.97	-0.14	15.81	54.7	0	21.67	45.71
r101	-1.64	28.57	-9.38	-5.25	30	13.71	-5.68	31.25	2.4	-3.68	27.78	-8.06
r102	-1.4	16.67	14.75	0	25	31.76	-0.6	18.75	18.93	-1.49	25	14.87
r103	-0.69	0	23.49	-1.95	33.33	21.1	-1.67	14.29	33.39	-3.38	36.36	32.55
r104	0.66	-25	9.58	-2.58	0	27.71	-2.74	26.67	21.42	-2.73	-6.25	13.14
r105	-3.64	40	6.59	-4.52	30	10.53	-0.67	31.25	1.7	-5.44	27.27	37.69
r106	-4.78	16.67	9.38	-3.63	16.67	22.27	-0.71	0	-12.41	-4.09	15	25.58
r107	-1.7	-20	26.54	-0.19	0	18.71	0.13	7.14	12.78	-0.55	15.79	25.52
r108	-1.62	0	21.93	-0.72	0	20.21	-3.15	7.14	22.8	-0.83	5.56	17.45
r109	-6.16	57.14	15.18	-5.14	41.67	32.95	-2.87	25	34.34	-2.29	27.27	8.37
r110	0	0	18.05	-0.39	12.5	7.61	-1.53	18.75	8.74	-1.37	25	9.69
r111	0.68	33.33	25.63	0	0	33.44	-0.79	14.29	22.59	-2.06	20	32.29
r112	-3.39	50	28.65	-1.3	22.22	26.24	-1.45	7.14	30.79	-1.69	15	20.6
Average	-1.97	16.45	15.87	-2.14	17.62	22.19	-1.81	16.81	16.46	-2.47	19.48	19.14
r201	-39.44	56.52	76.58	-28.71	58.7	79.29	-17.14	49.12	64.64	-8.16	48.57	36.74
r202	-11.56	52.38	75.38	-14.36	54.55	65.21	-6.86	53.45	57.5	-4.53	36.21	52.92
r203	-7.02	54.55	76.31	-9.4	50	57.06	-4.87	44.44	52.81	-0.14	42.86	46.27
r204	-0.85	37.5	78.91	-5.15	40.63	56.16	-0.96	40	55.23	0	25	51.52
r205	-28.84	64.29	82.33	-16.35	48.72	70.29	-3.7	51.67	52.43	-0.48	49.21	46.7
r206	-14.76	54.55	77.49	-7.97	51.22	72.61	-2.06	50	48.92	0	40	28.51
r207	-13.41	50	76.48	-3.91	54.55	51.84	-0.34	53.45	58.34	0	37.04	49.59
r208	-2.03	41.18	79.33	-2.67	44.44	64.19	0	38.46	56.43	0	9.68	43.52
r209	-22.07	56.52	77.05	-13.36	52.5	74.79	-3.36	46.15	58.49	0	30.61	48.55
r210	-16.19	52.38	79.51	-7.38	51.22	67.66	-2.61	45.1	58.01	0	48.39	53.81
r211	-15.61	41.18	77.47	-8.48	48.57	73.37	0	38.64	59.75	0	33.33	43.3
Average	-15.62	51	77.89	-10.7	50.46	66.59	-3.81	46.41	56.6	-1.21	36.45	45.58
rc101	0	0	-1.28	-1.87	-14.29	13.33	0	0	8.47	-0.39	15.38	-26.15
rc102	2.7	20	17.35	1.84	12.5	39.75	-0.43	-9.09	10.89	-2.62	-7.69	31.54
rc103	1.14	0	17.39	0.58	0	26.72	-4.46	8.33	28.2	-3.52	0	34.24
rc104	0	0	17.29	-3.31	12.5	14.51	-1.47	9.09	-2.15	-0.98	18.75	18.19
rc105	-1.23	0	7.53	-9	25	7.18	-2.16	18.18	10.56	-1.45	6.67	15.79
rc106	0	0	9.47	-3.53	0	31.5	-1.9	15.38	8.21	-1.62	7.14	23.47
rc107	-5.43	20	7.27	-5.25	12.5	1.16	-3.72	26.67	37.94	0.74	20	-5.76
rc108	-4.86	20	11.2	-0.19	20	27.03	-2.18	8.33	-1.32	-1.32	11.11	-8.14
Average	-0.96	7.5	10.78	-2.59	8.53	20.15	-2.04	9.61	12.6	-1.4	8.92	10.4
rc201	-16.86	47.37	68.49	-23.01	48.65	65.1	-16.73	49.06	59.28	-7.95	47.69	46.29
rc202	-6.49	28.57	69.61	-17.49	51.35	64.19	-13	46.94	53.4	-4.87	40.35	31.56
rc203	-5.75	44.44	76.93	-12.68	40.63	69.49	-6.5	41.86	36.7	-0.29	32.65	24.92
rc204	1.17	9.09	74.64	-3.54	26.92	67.54	-1.33	35	58.67	0	17.5	45.6
rc205	-18.93	33.33	68.28	-23.33	50	64.53	-17.11	42.86	62.31	-7.6	44.64	53.88
rc206	-13.58	44.44	75.35	-10.22	41.18	53.03	-8.02	44.68	59.28	0	52.31	45.09
rc207	-13.41	33.33	73.41	-5.37	38.71	68.18	-5.24	47.83	53.31	-0.35	50.82	33.93
rc208	-5.36	37.5	71.27	-1.68	34.48	63.33	0	44.9	44.77	0	41.51	50.52
Average	-9.9	34.76	72.25	-12.17	41.49	64.42	-8.49	44.14	53.47	-2.63	40.93	41.47
Total Av.	-5.58	25.32	42.19	-5.3	27.32	42.41	-2.91	25.98	32.67	-1.66	23.95	28.13

Table 7: Comparison between CSCRatio and CSCRoutes for Cordeau et al. instances

	1 tour Gap(%)			**2 tours Gap(%)**			**3 tours Gap(%)**			**4 tours Gap(%)**		
Name	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time	Profit	Transf	Time
pr01	-22.37	50	42.86	-11.36	53.85	47.17	0	25	-1.12	-0.76	41.18	31.65
pr02	-3.86	50	46.92	-5.68	50	48.56	-6.92	36.84	42.27	-7.16	20	36.61
pr03	-11.2	45.45	45.97	-10.94	47.62	60.57	-5.13	34.62	27.82	-7.96	19.23	50.64
pr04	-8.41	22.22	50.11	-4.21	20	50.71	-4.58	25	24.23	-7.57	27.03	16.39
pr05	-19.2	41.67	55.32	-12.32	26.09	60.42	-5.35	50	44.14	-4.91	20.41	38.86
pr06	-14.8	43.75	41.3	-9.85	29.17	44.92	-9.83	17.65	39.73	-9.76	33.33	12.52
pr07	0	0	40.15	-0.54	20	54.49	-3.88	37.5	43.01	-4.08	40	11.28
pr08	-10.99	0	47.77	-7.79	41.18	51.15	-10.7	28.57	54.43	-5.74	32.35	55.54
pr09	-5.56	37.5	39.03	-8.21	50	38.99	-11.31	29.63	53.08	-5.71	34.78	37.88
pr10	-6.82	50	47.05	-11.91	30.77	44.38	-9.27	24.39	45.67	-9.58	22	45.65
pr11	-5.59	62.5	11.34	1.34	40	38.07	-2.83	18.18	39.17	0	23.53	10.68
pr12	-5.99	40	51.88	-5.64	62.5	40.42	-3.02	37.5	60.84	0.19	23.81	52.87
pr13	-5.82	25	44.37	-6.13	52.94	48.02	-5.84	31.82	48.96	-7.81	38.24	44.99
pr14	-7.92	-33.33	54.59	-8.06	0	47.08	-2.42	31.25	40.95	-7.71	32.43	50.71
pr15	-6.6	50	65.07	-10.9	50	53.26	-3.45	38.24	42.93	-8.14	41.46	40.76
pr16	-9.01	45.45	54.19	-13.66	41.67	52.83	-9.73	20.69	35.22	-6.95	36.59	49.79
pr17	-5.16	42.86	52.06	-4.19	30	36.18	-1.51	7.69	38.99	-0.91	27.78	34.73
pr18	-21.99	50	53.12	-11.43	47.62	61.08	-7.33	29.17	45.93	-4.77	20.69	35.79
pr19	-4.51	0	42.55	-4.89	33.33	51.03	-6.76	27.59	41.84	-9.11	20.59	35.87
pr20	-12.27	14.29	54.98	-14.41	37.5	47.19	-6.22	21.21	35.66	-4.45	23.91	23.63
Average	-9.4	31.87	47.03	-8.04	38.21	48.83	-5.8	28.63	40.19	-5.64	28.97	35.84

Table 8: Comparison between ILS and CSCRatio for new instances

t1				**t2**			
Name	Profit Gap(%)	Transfer Gap(%)	Time Gap(%)	Name	Profit Gap(%)	Transfer Gap(%)	Time Gap(%)
t101	0	11.11	9.74	t201	2.19	0	-40.79
t102	-1.17	-10	-52.61	t202	0	0	-8.82
t103	2.16	-30	2.89	t203	2.87	-33.33	-79.1
t104	1.49	-11.11	9.71	t204	0	-66.67	17.65
t105	-1.15	12.5	-16.72	t205	-2.91	0	75.05
t106	1.03	0	-37.82	t206	0.51	-133.33	22.17
t107	-3.56	20	35.88	t207	0	0	-13.48
t108	-0.42	6.25	28.01	t208	8.64	0	-53.33
t109	-1.53	12.5	13.92	t209	1.98	0	45.35
t110	1.24	-7.14	24.23	t210	0.21	18.18	51.47
t111	-1.1	6.67	31.87	t211	0.21	-14.29	46.8
t112	-0.88	7.14	-29.86	t212	1.08	-25	34.01
t113	-2.38	10.53	39.55	t213	0.4	-27.27	21.94
t114	1.93	0	-22.46	t214	3.87	-33.33	25.53
t115	0.28	-16.67	-18.06	t215	0.24	-10	16.55
t116	0.12	-7.14	-41.73	t216	-0.22	9.09	40.5
t117	-1.33	33.33	12	t217	0	12.5	70.52
t118	-0.61	30.43	-19.2	t218	0	0	-38
t119	-1.81	16	-39.49	t219	1.9	-36.36	16.16
t120	-0.88	-6.67	32.77	t220	3.89	20	59.97
t121	4.95	25	-59.27	t221	2.5	10	9.52
t122	0.21	0	40.21	t222	0	-20	12.31
t123	1.49	-40	-37.25	t223	18.03	-100	-143.08
t124	7.36	-40	-87.22	t224	1	-20	44.16
t125	-0.6	36.84	5.13	t225	1.47	7.14	60.07
t126	0.24	-20	-8.5	t226	-1.05	0	35.62
t127	-0.2	13.33	27.77	t227	0	0	-36.99
t128	-0.54	0	-1.65	t228	-1.12	0	14.16
t129	2.31	14.29	-25.1	t229	0	0	-27.64
t130	-0.99	16.67	-23.22	t230	-0.69	-57.14	16.48
t131	1.75	-28.57	-23.29	t231	0.6	0	-19.01
t132	-0.95	30	-2.41	t232	2.49	26.67	50.95
t133	0.75	-41.67	38.78	t233	16.11	-20	-29.46
t134	1.24	0	40.31	t234	2.66	0	40.77
t135	-5.22	0	54.44	t235	2.48	-8.33	21.79
t136	0.79	-37.5	-24.56	t236	-0.57	0	-44.05
t137	-2.68	-11.11	22.01	t237	0.85	10	48.74
t138	-1.55	5.26	12.53	t238	-0.76	0	33.65
t139	3.5	0	4.79	t239	0.2	20	54.73
t140	4.23	27.78	-58.05	t240	3.7	-33.33	-3.83
t141	3.04	-15.38	-2.71	t241	1.18	-33.33	-104.17
t142	-0.59	25	-30.72	t242	0	0	-2.25
t143	0.73	0	-74.8	t243	14.71	-150	-76.62
t144	-3.93	-7.14	20.54	t244	0	11.11	-23.86
t145	3.64	33.33	-47.45	t245	2.75	0	19.05
t146	0.78	15.38	10.21	t246	-0.44	27.27	16.24
t147	2.32	6.67	38.51	t247	2.02	10	36.43
t148	1.5	0	-65.23	t248	4.94	0	13.18
t149	-0.65	23.81	40.58	t249	1.62	-20	52.32
t150	-0.41	0	-10.69	t250	0.5	0	36.96
Average	0.28	2.19	-5.27	Average	2	-13.2	8.33

Table 9: Comparison between ILS and CSCRoutes for new instances

t1				**t2**			
Name	Profit Gap(%)	Transfer Gap(%)	Time Gap(%)	Name	Profit Gap(%)	Transfer Gap(%)	Time Gap(%)
t101	-3.1	22.22	37.08	t201	-3.28	33.33	-34.21
t102	-0.78	-10	23.27	t202	0	0	-8.82
t103	1.4	-20	31.52	t203	2.3	33.33	-41.79
t104	0.27	0	39.68	t204	0	0	10.46
t105	0	0	-14.89	t205	-0.45	0	67.76
t106	-0.26	0	18.91	t206	0	-100	5.19
t107	-1.78	6.67	44.15	t207	15.52	-50	-24.72
t108	0.14	0	39.22	t208	8.64	0	-74.44
t109	-4.04	0	46.24	t209	-1.98	30.77	52.48
t110	0.25	-7.14	15.66	t210	3.33	18.18	38.67
t111	-0.73	0	37.67	t211	0.42	0	34.28
t112	2.63	0	-17.16	t212	-0.22	-25	40.61
t113	-3.02	10.53	54.56	t213	-1.61	18.18	17.45
t114	-2.36	-14.29	17.07	t214	2.26	0	34.47
t115	-2.27	-8.33	43	t215	0	-20	11.19
t116	-0.12	21.43	-14.74	t216	1.08	0	45.33
t117	2.21	11.11	29.52	t217	-0.22	12.5	58.34
t118	0	8.7	-6.87	t218	0	-100	-40
t119	-0.34	20	-27.83	t219	2.11	-9.09	9.07
t120	-2.05	6.67	64.23	t220	-1.5	0	68.75
t121	0.94	50	-9.6	t221	0	40	-2.54
t122	0.43	50	67.78	t222	-1.77	-10	15.19
t123	1.24	-60	-18.62	t223	25.14	-50	-149.23
t124	8.28	20	-23.33	t224	0.75	0	47.68
t125	0.26	26.32	32.29	t225	1.1	14.29	47.94
t126	-1.21	20	29.25	t226	1.58	7.14	28.84
t127	-1.27	6.67	46.96	t227	0	0	-57.53
t128	-4.41	-4.55	36.16	t228	-1.3	0	22.91
t129	2.08	0	0	t229	-2.25	25	-43.09
t130	-5.91	33.33	-7.01	t230	-1.04	-14.29	-9.66
t131	-8.75	0	18.32	t231	-1.81	0	19.48
t132	-0.48	30	7.85	t232	0.57	13.33	49.96
t133	0.75	-16.67	52.69	t233	18.33	-40	-45.54
t134	1.73	12.5	37.54	t234	3.07	-25	51.13
t135	-2.67	0	61.21	t235	-0.41	8.33	8.12
t136	-1.32	-25	9.21	t236	0	0	-57.14
t137	-4.47	11.11	36.56	t237	1.27	0	48.53
t138	-1.31	5.26	11.35	t238	1.33	12.5	39.95
t139	2.06	15	33.54	t239	0	13.33	46.33
t140	1.11	27.78	14.29	t240	8.42	-33.33	-26.78
t141	0.83	-7.69	35.84	t241	1.18	0	-135.42
t142	-1.6	8.33	24.21	t242	0	0	6.74
t143	0.73	-12.5	-49.21	t243	18.24	-100	-93.51
t144	-4.46	7.14	42.64	t244	0.3	22.22	-3.98
t145	1.96	33.33	22.96	t245	-2.06	-40	-8.33
t146	0.13	-7.69	38.96	t246	-0.66	18.18	20.44
t147	0.65	-33.33	39.36	t247	-0.22	10	27.49
t148	1.5	0	2.34	t248	4.49	-7.69	33.16
t149	1.12	28.57	41.35	t249	-1.39	40	51.27
t150	0	0	16.35	t250	-4	28.57	26.81
Average	-0.52	5.31	22.23	Average	1.91	-4.5	4.59

Table 10: Comparison between CSCRatio and CSCRoutes for new instances

t1				**t2**			
Name	Profit Gap(%)	Transfer Gap(%)	Time Gap(%)	Name	Profit Gap(%)	Transfer Gap(%)	Time Gap(%)
t101	-3.1	12.5	30.29	t201	-5.35	33.33	4.67
t102	0.39	0	49.72	t202	0	0	0
t103	-0.75	7.69	29.48	t203	-0.56	50	20.83
t104	-1.2	10	33.2	t204	0	40	-8.73
t105	1.17	-14.29	1.56	t205	2.53	0	-29.21
t106	-1.27	0	41.16	t206	-0.51	14.29	-21.82
t107	1.84	-16.67	12.91	t207	15.52	-50	-9.9
t108	0.56	-6.67	15.56	t208	0	0	-13.77
t109	-2.55	-14.29	37.55	t209	-3.88	30.77	13.04
t110	-0.98	0	-11.31	t210	3.11	0	-26.38
t111	0.37	-7.14	8.51	t211	0.21	12.5	-23.53
t112	3.53	-7.69	9.78	t212	-1.29	0	10
t113	-0.66	0	24.82	t213	-2	35.71	-5.75
t114	-4.2	-14.29	32.27	t214	-1.55	25	12.01
t115	-2.54	7.14	51.72	t215	-0.24	-9.09	-6.42
t116	-0.24	26.67	19.05	t216	1.3	-10	8.12
t117	3.59	-33.33	19.91	t217	-0.22	0	-41.32
t118	0.62	-31.25	10.34	t218	0	-100	-1.45
t119	1.49	4.76	8.36	t219	0.21	20	-8.46
t120	-1.18	12.5	46.8	t220	-5.19	-25	21.94
t121	-3.82	33.33	31.19	t221	-2.44	33.33	-13.33
t122	0.21	50	46.1	t222	-1.77	8.33	3.29
t123	-0.24	-14.29	13.57	t223	6.02	25	-2.53
t124	0.86	42.86	34.12	t224	-0.25	16.67	6.31
t125	0.86	-16.67	28.62	t225	-0.36	7.69	-30.4
t126	-1.45	33.33	34.8	t226	2.66	7.14	-10.53
t127	-1.08	-7.69	26.56	t227	0	0	-15
t128	-3.89	-4.55	37.2	t228	-0.19	0	10.19
t129	-0.23	-16.67	20.06	t229	-2.25	25	-12.1
t130	-4.98	20	13.16	t230	-0.35	27.27	-31.29
t131	-10.32	22.22	33.75	t231	-2.4	0	32.35
t132	0.48	0	10.02	t232	-1.87	-18.18	-2.02
t133	0	17.65	22.72	t233	1.91	-16.67	-12.41
t134	0.49	12.5	-4.64	t234	0.4	-25	17.49
t135	2.69	0	14.87	t235	-2.82	15.38	-17.49
t136	-2.1	9.09	27.11	t236	0.57	0	-9.09
t137	-1.84	20	18.66	t237	0.42	-11.11	-0.41
t138	0.25	0	-1.35	t238	2.11	12.5	9.48
t139	-1.39	15	30.2	t239	-0.2	-8.33	-18.55
t140	-3	0	45.77	t240	4.55	0	-22.11
t141	-2.14	6.67	37.54	t241	0	25	-15.31
t142	-1.02	-22.22	42.02	t242	0	0	8.79
t143	0	-12.5	14.64	t243	3.08	20	-9.56
t144	-0.55	13.33	27.81	t244	0.3	12.5	16.06
t145	-1.62	0	47.75	t245	-4.68	-40	-33.82
t146	-0.65	-27.27	32.02	t246	-0.22	-12.5	5.01
t147	-1.63	-42.86	1.38	t247	-2.2	0	-14.05
t148	0	0	40.9	t248	-0.43	-7.69	23.01
t149	1.78	6.25	1.3	t249	-2.97	50	-2.21
t150	0.41	0	24.43	t250	-4.48	28.57	-16.09
Average	-0.78	1.46	24.48	Average	-0.12	4.85	-5.25

3.6 Conclusions

We introduced CSCRatio and CSCRoutes, two cluster-based approaches to the TTDP. The main design objectives of the two algorithms address the main shortcomings of the best known so far real-time TTDP algorithm, ILS. The main incentive behind our approaches is to favor visits to topology areas featuring high density of good candidate vertices. Furthermore, they both favor solutions with reduced number of long transfers among vertices, which are associated with public transportation transfers in typical urban settings (such transfers are costly, time consuming and usually less attractive to tourists than short walking transfers).

The comparison of CSCRatio over the best known real-time TTDP algorithm (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 comparison of CSCRoutes over ILS, this confirmed the prevalence of the former 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. Furthermore, 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 vertices typically feature wide, overlapping time windows and are located nearby each other, while the daily time budget is 5-10h.

Based on the above findings, our two cluster-based heuristics may be thought of as complementary TTDP algorithmic options. We argue that 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.

4 Modeling the Tourist Trip Design Problem as a Time-Dependent Team Orienteering Problem with Time Windows

4.1 Introduction

Tourists visiting urban destinations typically deal with the challenge of making a feasible plan in order to visit the most interesting attractions (POIs) in their available time span. The filtering of most important POIs (amongst the many available) and their time-sequencing along the tourist itineraries is a particularly cumbersome task [6, 45].

The situation is further complicated when considering the complexity of metropolitan transit networks commonly used by tourists to move from a POI to another, whenever walking is not an option, due to distance constraints. Tourists are typically unfamiliar with and intimidated by the nuances of the public transit systems in their destination areas [41], thereby making transit transfers a complicated exercise. Tourists are especially reluctant in using bus networks as they feel they do not have the acquired local knowledge to negotiate them efficiently, while also running the risk of leaving the tourism space and entering a terra incognita, should they use a wrong service or take a wrong direction [33].

An interesting aspect highlighted by field studies is that tourists are 'outcome' oriented and seek to maximize time spent at a place by minimizing transit time. Unlike commuters, most tourists would trade a time-efficient transit transfer in favor of a more indirect, scenic or roundabout walking route that offers more opportunities for amorphous exploration and discovery [39]. Many tourists opt for public transportation when pedestrian walking is long enough to challenge their strength and endurance. Even then, any delays incurred on stops (waiting to board on the next transiting service) are highly undesirable, given the limited time budget spent on the tourist destination [33].

The above discussion underlines the need for ICT tools (i.e. TTDP solvers), to assist the way arounds of tourist transfers among POIs, either walking or using public transit. Notably, most algorithmic approaches addressing TTDP assume constant travel times among POIs (this is equally true for our cluster-based TOPTW algorithms introduced in Section 3.4), 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 [33]. The only approach that deviates from this unrealistic TTDP viewpoint is the one proposed by Garcia et al. [21] which integrates the option for multimodal transit transfers in the TTDP modeling. However, the proposed algorithm design is based on the simplified assumption of periodic service schedules; the latter, clearly, is not valid in realistic complex transportation networks, wherein arrival/departure frequencies typically vary within the services operational periods while deviations from planned service time schedules commonly occur due to non-predictable events. Moreover, this algorithm only considers predefined start/end locations for tourist routes.

Herein, we propose two novel heuristic algorithmic approaches, the Time Dependent CSCRoutes (TDCSCRoutes) and the SlackCSCRoutes, which address the above described shortcomings of existing approaches to TTDP. The main incentive behind our approaches is to motivate visits to topology areas featuring high density of 'good' (i.e. highly profitable) candidate vertices, while taking into account time dependency (i.e. multimodality) in calculating travel times from one vertex 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.

Both heuristics involve a precalculation phase, wherein POIs are grouped in disjoint clusters, based on geographical criteria; thus, any pair of POIs that belong to the same cluster is likely to be within walking distance. This phase also involves the offline calculation of time-dependent travel times among a fixed set of nodes, based on provided transit timetable data. The nodes' set comprises POIs and accommodation locations; the latter are considered as potential start/end locations of the daily tourist itineraries. In the online phase, TDCSCRoutes and SlackCSCRoutes take a local and a global criterion, respectively, for inserting POIs along the recommended routes. The two algorithms favor solutions with increased number of walking over public transit transfers (the latter are considered costly and typically less attractive to tourists than short walking transfers). We also propose extensions on our algorithms that tackle the 'Generalized TTDP', which involves arbitrary (i.e. determined at query time) rather than fixed start/end locations for derived tourist itineraries.

In addition to TDCSCRoutes and SlackCSCRoutes, we have also implemented another algorithm (AvgCSCRoutes) which uses average (rather than time dependent) travel times among POIs. That way, AvgCSCRoutes effectively reduces TDTOPTW to TOPTW. Having obtained a TOPTW solution, AvgCSCRoutes employes two further steps to ensure route feasibility and improve the solution's quality.

Our prototyped algorithms have been tested in terms of various performance parameters (solutions quality, execution time, percentage of transit transfers over total transfers, etc) upon real test instances (i.e. set of POIs and accommodation facilities) 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 a time dependent extension of ILS as well as two variants that use precalculated average travel times (among the individual time dependent, real travel times) between POIs.

The remainder of this Section is organized as follows: Section 4.2 presents our novel cluster-based heuristics, while Section 4.3 details our approach in solving the Generalized TTDP. Section 4.4 describes a method used for preprocessing multimodal travel time distances among POIs. Section 4.5 discusses the experimental results compiled from executing our algorithms upon real test instances. Finally, Section 4.6 concludes our work and suggests directions for future work.

4.2 TDTOPTW heuristics

TDTOPTW is an extension of TOPTW integrating public transportation, i.e., time dependent travel costs among nodes. Hence, we use the same mathematical notation, as introduced in Section 3.2. In TDTOPTW, every link $(u, v) \in E$ (where E is the set of edges in the directed graph $G = (V, E)$) denotes the transportation link from u to v and is assigned a travel time. The objective is to find k disjoint routes each starting from a starting location $s \in N$ and ending at a location $t \in N$, each with overall duration limited by the time budget B_r, 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).

Figure 8 depicts a typical tourist route from a start to an end location via a series of POIs, each associated with a time window and a required time to visit. Each transit transfer among two locations is subject to a delay (e.g. t_3 - t_2 when leaving from p_i to p_j). Such delays do not occur when taking the walking transfer option rather than public transit (e.g. transfer from p_k to t_r). Each visit is also likely to be delayed if the tourist arrives at a POI before its opening hour (e.g. waiting of t_5 - t_4 prior to start visiting p_i). A TDTOPTW solver should minimize the overall delays incident along the routes and and exploit the time saved in order to accommodate visits to additional POIs.

Figure 9 illustrates the arrival time at a POI p_j for a tourist that has previously visited p_i. Delays incurred to embark on the next service may vary, i.e. we make no assumption of periodic service schedules. Also, when considering certain departure times (e.g. between t_0 and t_1) walking might be a preferable option than waiting for the next transit service.

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 (see Section 3.4.2) to handle time dependent travel times among different

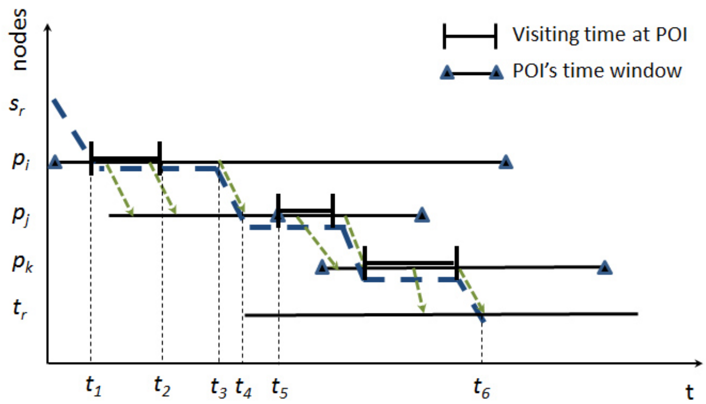


Figure 8: Illustration of a tourist route (blue dashed line) from s_r to t_r via POIs p_i, p_j and p_k. Green dashed arrows indicate available multimodal transfer options among POIs.

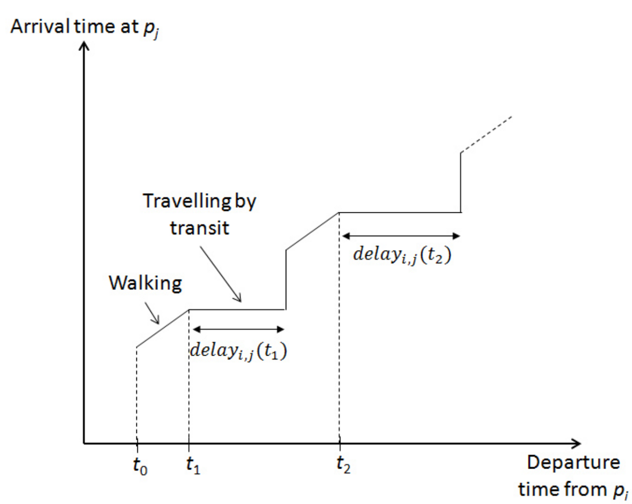


Figure 9: Arrival time walking and using public transportation

locations/POIs. Recall that 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 [50].

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 4.2.1 we present the feasibility criterion for inserting a POI p in a route r in the case of time dependent travel costs. In the following three subsections we describe three algorithmic approaches for solving the TDTOPTW problem namely, the TDCSCRoutes algorithm, the SlackCSCRoutes algorithm and the AvgCSCRoutes algorithm.

4.2.1 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. To determine all the non-dominated pairs in S_{uv} we employ the algorithm of Dibbelt et al. [13].

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.,

$$\text{deptime}_{u,v}(t) = \min_i \{\text{dep}_i^{uv} | (\text{dep}_i^{uv}, \text{trav}_i^{uv}) \in S_{uv} \text{ and } t \leq \text{dep}_i^{uv}\} \tag{10}$$

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 $\text{delay}_{u,v}(t) = \text{deptime}_{u,v}(t) - t$. For instance, in Figure 10, $\text{deptime}_{u,v}(t_1) = \text{dep}_1^{uv}$. Therefore, the total travelling cost from u to v at a specified time t, $\text{travelling}_{u,v}(t)$, is

$$\text{travelling}_{u,v}(t) = \min\{\text{walking}_{u,v}, \text{delay}_{u,v}(t) + \text{travtime}_{u,v}(t)\} \tag{11}$$

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 $\text{maxStart}_{t_r} = et_r$.

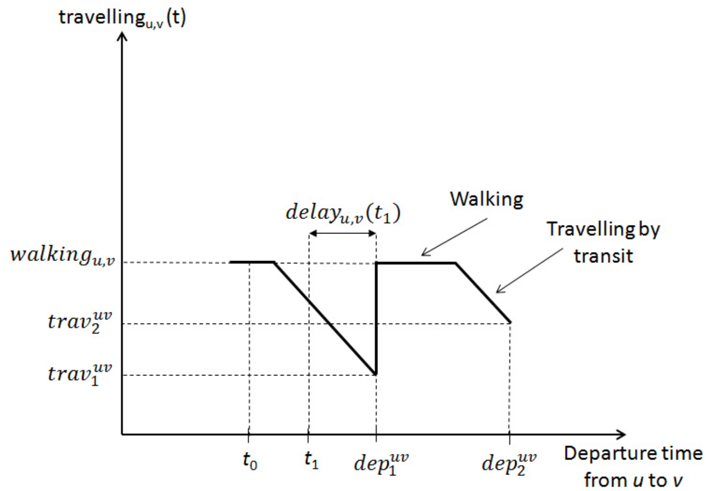


Figure 10: Travelling time from u to v as a function of the departure time from u

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.

$$\text{shift}_k^{ij} = (\text{travelling}_{i,k}(\text{leave}_i) + \text{wait}_k + \text{visit}_k + \text{travelling}_{k,j}(\text{leave}_k)) - \text{travelling}_{i,j}(\text{leave}_i) \quad (12)$$

Figure 11 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).

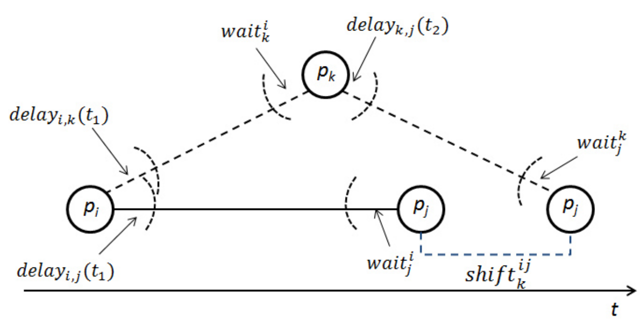


Figure 11: Illustration of p_k insertion between p_i and p_j.

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

$$\text{arrive}_k \leq \text{close}_{kr} \ \text{ and } \ \text{shift}_k^{ij} \leq \text{maxStart}_j - \text{arrive}_j \quad (13)$$

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

Algorithm 5 shift(k, i, j, r)

result $\leftarrow \infty$
arrive$_k \leftarrow$ leave$_i$ + travelling$_{i,k}$(leave$_i$)
if arrive$_k \leq$ close$_{kr}$ **then**
  wait$_k \leftarrow \max(0, \text{open}_{kr} - \text{arrive}_k)$
  leave$_k \leftarrow$ arrive$_k$ + wait$_k$ + visit$_k$
  costAfterInsert $\leftarrow$ travelling$_{i,k}$(leave$_i$) + wait$_k$ + visit$_k$ + travelling$_{k,j}$(leave$_k$)
  shift$_k^{ij} \leftarrow$ costAfterInsert $-$travelling$_{i,j}$(leave$_i$)
  if shift$_k^{ij} \leq$ maxStart$_j$ $-$ arrive$_j$ **then**
    result $\leftarrow$ shift$_k^{ij}$
  end if
end if
return result

4.2.2 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_C^r. 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 $CR_{C'}^r$. 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_C^r for some cluster C. Therefore, a POI cannot be inserted at any position in the route r [23].

In the sequel, the description of insertion step of TDCSCRoutes (**TDCSCRoutes_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_f^r be the first cluster route (starting at s_r) in r, and CR_l^r 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_C^r, 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_f^r and CR_l^r
  - 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_l^r
  - 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 $CR_{\text{cluster}(p_k)}^r$
- cluster$(s_r) \neq$ cluster(t_r)
  - if cluster(p_k) = cluster(s_r) then p_k can be inserted everywhere in CR_f^r
  - if cluster(p_k) = cluster(t_r) then p_k can be inserted everywhere in CR_l^r

– if $\text{cluster}(p_k) \in \text{clustersIn}(r)$ and $\text{cluster}(p_k)$ is different from both $\text{cluster}(s_r)$ and $\text{cluster}(t_r)$, then p_k can be inserted everywhere in $CR^r_{\text{cluster}(p)}$

– if $\text{cluster}(p_k) \notin \text{clustersIn}(r)$ 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

$$\text{ratio}^{ij}_k = \frac{\text{profit}^2_k}{\text{shift}^{ij}_k}(1 + a \cdot \frac{D^{ij}_k + 1}{D^{ij}_k + 2} + (1-a) \cdot f(\text{shift}^{ij}_k, \text{wait}_j + \text{delay}_j)) \tag{14}$$

where $f(x, y) = 1$ if $x \leq y$ and 0 otherwise, and $D^{ij}_k = \text{delay}_{i,k}(\text{leave}_i) + \text{wait}_k + \text{delay}_{k,j}(\text{leave}_k) + \text{wait}_j$ 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 [23]. The incentive behind (14) is the following: $\frac{\text{profit}^2_k}{\text{shift}^{ij}_k}$ denotes preference for important (i.e. highly profitable) POIs associated with relatively short time to visit. In the first iterations (a=1), the operand $\frac{D^{ij}_k+1}{D^{ij}_k+2}$ 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), $f(\text{shift}^{ij}_k, \text{wait}_j + \text{delay}_j)$ 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. The variables of p_k are updated as follows:

$$\begin{aligned}
&\text{arrive}_k = \text{leave}_i + \text{travelling}_{i,k}(\text{leave}_i)\\
&\text{wait}_k = \max(0, \text{open}_{kr} - \text{arrive}_k)\\
&\text{start}_k = \text{arrive}_k + \text{wait}_k\\
&\text{leave}_k = \text{arrive}_k + \text{wait}_k + \text{visit}_k\\
&\text{maxStart}_k = \min(\text{close}_{kr}, \max\{t : t + \text{travelling}_{k,j}(t) \leq \text{maxStart}_j\} - \text{visit}_k)
\end{aligned}$$

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, recursively computed as follows:

$$\text{maxStart}_l = \min(\text{close}_{lr}, \max\{t : t + \text{travelling}_{l,next(l)}(t) \leq \text{maxStart}_{next_l}\} - \text{visit}_l) \tag{15}$$

The pseudo code of **TDCSCRoutes_Insert** is listed below (Algorithm 6).

4.2.3 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 (see Figure 12) defined for each node p_i in a tourist route r as follows:

$$\text{slack}_i = \text{maxStart}_i - \text{arrive}_i \tag{16}$$

Algorithm 6 TDCSCRoutes_Insert

```
for each candidate POI p_k do
    for each route r do
        if cluster(s_r)= cluster(t_r) then
            if clustersIn(r)= {cluster(s_r)} then
                Search all positions in r for the highest ratio
            else// clustersIn(r)≠ {cluster(s_r)}
                clusterID←cluster(p_k)
                if clusterID=cluster(s_r) then
                    Search all positions in CR^t_f and CR^t_l for the highest ratio
                else// clusterID≠cluster(s_r)
                    if clusterID ∉ clustersIn(r) then
                        Search all positions in r that are the end of a CR, for the highest ratio
                    else// clusterID ∈ clustersIn(r)
                        Search all positions in CR^r_clusterID for the highest ratio
                    end if
                end if
            end if
        else// cluster(s_r)≠ cluster(t_r)
            clusterID←cluster(p_k)
            if clusterID = cluster(s_r) then
                Search all positions in CR^r_f for the highest ratio
            else// clusterID ≠ cluster(s_r)
                if clusterID = cluster(t_r) then
                    Search all positions in CR^r_l for the highest ratio
                else//clusterID ≠ cluster(t_r)
                    if clusterID ∉ clustersIn(r) then
                        Search all positions in r that are the end of a CR, for the highest ratio
                    else// clusterID ∈ clustersIn(r)
                        Search all positions in CR^r_clusterID for the highest ratio
                    end if
                end if
            end if
        end if
    end for
end for
Insert the POI p_l with the highest ratio.
Update the variables of each POI in r and the set of cluster members for each cluster.
```

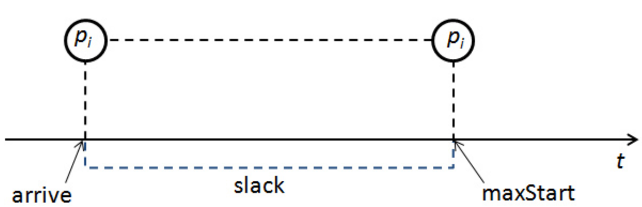


Figure 12: Illustration of slack's duration for POI p_i

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 two reasonable assumptions. The first one is that the time windows at the POIs are fairly long spanning the most part of the day and therefore the waiting time (wait_j) before each POI p_j ($j = 1 \ldots n$) is typically zero. This clearly holds for most tourist sites. We also assume that $\text{travelling}_{j,j+1}(\text{leave}_j) \approx \text{travelling}_{j,j+1}(\text{leave}_j^k)$, $j = i+1, \ldots n$, where leave_j^k is the new leave time of all nodes following the newly inserted node p_k along the route. The rational behind this approximation is that the additional delay caused by the new detour for visiting node p_k (i.e. shift_k^{ij}) is expected to be relatively short and so the time differences among subsequent POIs arrival times remains unaffected (intuitively, this removes the complexity associated with the time windows and time dependency). As a result, the same frequencies of public transport services still hold and so the travelling time between two successive nodes on the route can be considered the same as it was before the insertion. In order to quickly compute an approximation of A_k^i, we further assume for the moment that the departure delay at each POI due to public transport timetables is zero. Then, if $\text{shift}_k^{i(i+1)} = \text{travelling}_{i,k}(\text{leave}_i) + \text{wait}_k + \text{visit}_k + \text{travelling}_{k,i+1}(\text{leave}_k) - \text{travelling}_{i,i+1}(\text{leave}_i)$, it holds that

$$\text{arrive}_j^k - \text{arrive}_j \approx \text{Shift}_k^{i(i+1)}, \quad j = i+1 \ldots n$$

.

For reasons similar to those mentioned above, it will also hold that

$$\text{maxStart}_j - \text{maxStart}_j^k \approx \text{maxStart}_i - \text{maxStart}_i^k, \quad j = 1 \ldots i$$

In that case, if $sumL\_\text{slack}_i = \sum_{j=1}^{i} \text{slack}_j$ and ($sumR\_\text{slack}_i = \sum_{j=i+1}^{n} \text{slack}_j$), (that is the sum of slack parameters for the part of the trip from POI p_1 up to POI p_i and p_i up to POI p_n, respectively) as has been estimated in previous global iteration, the new A_k^i for inserting POI p_k will be

$$A_k^i = \frac{sumL\_\mathrm{slack}_i + i \cdot (\mathrm{maxStart}_i^k - \mathrm{maxStart}_i) + \mathrm{slack}_k + sumR\_\mathrm{slack}_i - (n-i) \cdot \mathrm{Shift}_k^{i(i+1)}}{n+1}$$

In the above derivations, we have disregarded the departure delays at each POI due to public transport timetables, e.g. the delays incurred when waiting on transit stops. Note that when the visit to a POI p_j $j = i+1, \ldots, n$ is delayed due to the insertion of POI p_k, we may be lucky and get the same bus, for instance, from POI p_{i+1} to p_{i+2} or unlucky and just miss the bus and wait for the next one. These two possibilities can happen at each subsequent trip leg. Since, the visit time at each POI can be considered random and the time each service arrives at a stop can be also considered random in our setting (for instance, the bus can arrive bus at 11.08 and not 11.05 or 11.10), we can assume that this time savings and losses due to the fixed transit timetable are canceling out along the trip. Thus, ignoring these delays in the above derivations may also be a good approximation.

4.2.4 The Average Travel Times CSCRoutes (AvgCSCRoutes) algorithm

In this subsection we discuss the AvgCSCRoutes which is based on the average travel time approach proposed by Garcia et al. [21] to handle time dependent travel costs among locations and integrate public transportation. For each pair of locations $u, v \in V$ the average travel cost ($\mathrm{avTravel}_{u,v}$) is precalculated using the time dependent travelling costs with time steps of one minute ($24 \cdot 60 = 1440$ time steps per day).

$$\mathrm{avTravel}_{u,v} = \frac{\sum_{r=1}^{7} \sum_{t=0}^{1439} \mathrm{travelling}_{u,v}^r(t)}{7 \cdot 1440}$$

where r represents the day of the week, and $\mathrm{travelling}_{u,v}^r(t)$ is the travelling cost from u to v at time t on the r^{th} day of the week. Then, for each POI p_i in a route r, the values of variables wait_i, start_i, leave_i and arrive_i are determined using the average travel costs among locations, while the value of $\mathrm{maxStart}_i$ is calculated as follows

$$\mathrm{maxStart}_i = \min(\mathrm{close}_{ir}, \mathrm{maxStart}_{\mathrm{next}(i)} - \mathrm{avTravel}_{i,\mathrm{next}(i)} - \mathrm{visit}_i)$$

Note that once the average travel times are available, the problem can be solved using a TOPTW algorithm, thereby removing time dependency. AvgCSCRoutes algorithm invokes CSCRoutes TOPTW subroutine. Certainly, the routes created by CSCRoutes will not take into account the real time dependent travel times between successive POIs. For this reason, the following update procedure is applied by the AvgCSCRoutes algorithm, to update the travel costs appropriately:

1. For each route $r = p_1, p_2, \ldots, p_l$, starting from the pair (p_1, p_2) and for each following pair (p_i, p_{i+1}) in r, $i = 2, \ldots, l-1$, the time dependent travel time $\mathrm{travelling}_{i,i+1}^r(\mathrm{leave}_i)$ is calculated using the set of non-dominated pairs $S_{p_i p_{i+1}}$.

2. If the time dependent travel time from p_i to p_j is shorter than the average one, then the visit at p_j starts earlier.In the opposite case, the visit at p_j (and, most likely, at some nodes following p_j) starts later. In both cases the variables of each POI in r are updated appropriately. Note that the above steps may violate the feasibility of one or more visits along a route r; in such case, the whole route r becomes infeasible..

3. In the case that one or more routes are infeasible, the following repair step is applied: While a route r is infeasible, the first, according to the visiting order, POI p_k in r with starting time (calculated based on the time dependent travel costs) greater than the starting time (calculated based on the average travel cost) is removed from r; the POIs following p_k in r are moved backwards and the proper arrival times are recalculated for these POIs. If p_k coincides with the end of the route, then the previous POI is removed.

4. At this point of the procedure, all routes in the solution are feasible, but there might exist "gaps" between POIs , allowing possible insertions of new POIs along the routes. Since the routes are almost "full", it seems that a good criterion for an insertion is to insert the highest profit POI in a position with the least shift (calculated based on the time dependent travel times). Thus, the last step of the procedure is as follows: Sort the POIs that do not belong to the routes of the solution in descending order of profit. Let L be the sorted list of POIs. Starting from the highest profit POI p_i and until the list L is empty do the following: if there exists one or more feasible insert position for p_i find one with the lowest shift over all routes and insert p_i; delete p_i from L and repeat.

4.3 Solving the Generalized Tourist Trip Design Problem

By solving the TTDP we expect to derive k routes each of length at most B, that maximize the overall collected profit. Each route may start and end at the tourist's accommodation location, or alternatively, at different user-defined starting and ending locations. TTDP may be formulated and solved as a TDTOPTW, where the POIs as well as the route starting and ending locations are formulated as nodes of the graph G (Section 4.2). In the sequel, we consider a generalization of the problem (Generalized TTDP) where the starting and the ending locations of a route may be any location in the destination city, i.e., they are both determined at runtime. This is in accordance with the typical envisaged usage scenario, whereby the TTDP solver will be inquired by a mobile client; the tourist's starting location will be typically fixed to his current position and the ending location will be also defined arbitrarily by the user at query time. Clearly, the formulation of the TDTOPTW problem using precalculated travel costs among a fixed set of predefined locations/nodes (e.g. POIs and hotels) cannot support the above described dynamic usage scenario. Therefore, the Generalized TTDP cannot be solved by the TDTOPTW algorithms presented in Section 4.2, and we need to further elaborate on an approach for its solution.

In the sequel we present an algorithm for solving the Generalized TTDP. The algorithm comprises a preprocessing phase and an on-line phase. The preprocessing phase consists of the following steps: First the global k-means clustering algorithm is applied on the set of POIs of the destination city and a set of clusters of POIs is constructed. Then the city is partitioned into small square regions (e.g. $500m \times 500m$), covering the whole geographical where POIs are located in. Within each region R_i a central location is chosen as the location that represents R_i, called region representative rep_i. Consider the complete directed graph $G = (V, E)$, where V consists of all the POIs and all region representatives. Then for each pair of locations (i, j) in V, the set S_{ij} of non-dominated pairs of departure and travel times is calculated (see Subsection 4.2.1). Finally, for each representative rep_i of a region i, consider that rep_i belongs to the nearest cluster based on the geometric distance of the mean of the POIs (i.e. centroid) of the cluster.

In the on-line phase of the algorithm, we assume that we are given a set of pairs (s_i, t_i), $i = 1, .., k$, denoting the starting and terminal locations of the i^{th} route, and a set of pairs (st_i, et_i), $i = 1, .., k$, denoting the starting and ending times of the i^{th} route. Then, the on-line phase of the algorithm proceeds as follows:

1. For each route r_i, $i = 1, .., m$, (i) find the representatives s'_i and t'_i of the regions where s_i and t_i belong to. (ii) Compute the distances $d_s = dist(s_i, s'_i)$ and $d_t = dist(t_i, t'_i)$. Note that due to the small size of the regions, d_s and d_t are walking distances; let t_{d_s} and t_{d_t} be the corresponding walking times. (iii) Set $st'_i = st_i + t_{d_s}$ and $et'_i = et_i - t_{d_t}$

2. Execute the TDCSCRoutes algorithm with input the new attributes s'_i, t'_i, st'_i, et'_i, for $i = 1, \dots, k$.

3. For each route $r_i = (s'_i, b_i = p_{i1}, p_{i2}, \dots, p_{ik} = l_i, t'_i)$ obtained by TDCSCRoutes, replace s'_i by s_i, st'_i by st_i, t'_i by t_i and et'_i by et_i. In the case that the walking time from s_i to b_i is shorter than the walking time from s_i to s'_i plus the travelling time from s'_i to b_i, the visit to b_i may

start earlier. Therefore, the variables of each POI in r_i should be updated accordingly. Also, in the case that the walking time from l_i to t_i is shorter than the the travelling time from l_i to t'_i plus the walking time from t'_i to t_i, the visit to l_i may start later. Therefore, the maxStart variables of each POI in r_i should be updated accordingly.

4. Note that after step 3, time "gaps" may appear between POIs along r_i. To fill these gaps, another step is applied as follows: Sort the POIs that do not belong to any route r_i, $i = 1, .., m$, in descending order of profit. Let L be the sorted list of POIs. Starting from the highest profit POI p_i and until the list L is empty do the following: if there exists one or more feasible insert position for p_i in any route r_i, $i = 1, ..., k$, find one with the lowest shift over all routes and insert p_i; delete p_i from L and repeat.

The pseudo code of the algorithm for solving the Generalized TTDP follows (Algorithm 7).

Algorithm 7 Generalized TTDP Algorithm

```
Preprocessing Phase
Cluster the POIs using global k-means
Partition the city into square regions. For each region R_i, choose a representative rep_i. Consider as locations the
representatives of the regions and the POIs
For each region representative rep_i consider that rep_i belongs to the nearest cluster
Calculate the time dependent travel times between all locations

On-line Phase
for each pair s_i, t_i do
    Find the representatives of s_i and t_i, rep_{s_i} and rep_{t_i}, respectively, and set s'_i = rep_{s_i} and t'_i = rep_{t_i}.
    Compute the walking distances d_s = d(s_i, s'_i) and d_t = d(t_i, t'_i)
    set st'_i = st_i + t_{d_s}
    set et'_i = et_i - t_{d_t}
end for
Execute the TDCSCRoutes with the new attributes (s'_i, t'_i, st'_i, et'_i), i = 1, ..., m
for each route r_i, r_i = (s'_i, b_i = p_{i1}, p_{i2}, ..., p_{ik} = l_i, t'_i) obtained by TDCSCRoutes do
    Replace s'_i by s_i, st'_i by st_i, t'_i by t_i and et'_i by et_i, and update the variables at each POI of r_i, if needed
end for
Sort the POIs that do not belong to any route r_i, i = 1, .., m, in descending order of profit; let p_1, ..., p_k be the
sorted list
for j = 1 to k do
    if there exists one (or more) feasible insert position for p_j in any route r_i, i = 1, ..., m, then
        Find the position with the lowest shift over all routes and insert p_j
    end if
end for
```

A typical solution to the Generalized TTDP is illustrated in Figure 13. The tourist destination area is partitioned in nine square regions and a central location (representative) is calculated for each region (indicated by green circles). The start/end locations of the route (s_1 and t_1, respectively) are determined at the user query time and are indicated by the black squares. The representatives s'_1 (of the area where s_1 belongs to) and t'_1 (of the area where t_1 belongs to) are visited in the beginning and in the end of the trip, respectively. POIs p_{11}, p_{12}, p_{13} and p_{14} are visited in between. Solid and dashed lines denote walking and transit transfers, respectively.

4.4 Preprocessing Multimodal Travel Time Distances

In their most-inner loop, algorithms for tourist trip design optimization require the pairwise travel time distance between points of interest (POIs). Typically in the literature such distances are assumed to be already available in a two-dimensional matrix quadratic in the number of POIs. In our scenario however, we consider multimodal travel times between POIs also dependent on the time of day. In this section we shortly present algorithms to precompute such a time-dependent distance matrix between POIs. This preprocessing step ensures fast travel time lookup during tourist trip

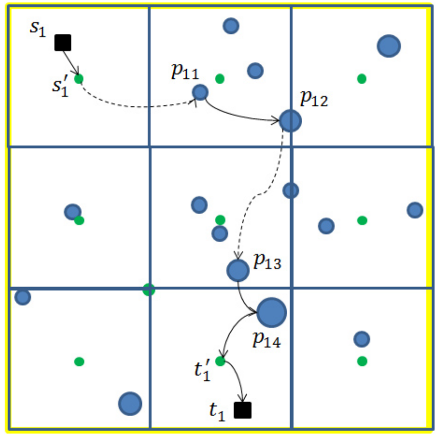


Figure 13: Illustration of a solution to the Generalized TTDP.

optimization. We evaluate the performance of our approach for POIs chosen in the multimodal network of the greater Athens area (see details in Section 4.5.1). A more detailed exposition can be found in [4, 12].

4.4.1 Multimodal Profile Queries

We consider the combined multimodal network of walking, taxi and public transit. Each subnetwork is modeled as a weighted graph $G_i(V_i, A_i)$, separately. For the *road networks* (e. g., walking, taxi), intersections are modelled by vertices $v \in V_i$, road segments as arcs $a \in A_i$; each arc is weighted by the expected travel time necessary to traverse it. *Public transit networks* are modeled following the time-dependent approach [42]; it assigns each arc a time-dependent weight function based on the departure and travel times of public transit connections, also accounting for waiting on the next trip. All subnetworks are combined into a multimodal graph $G(V, A), V = \bigcup V_i, A = \bigcup A_i \cup A_l$, adding link arcs $a \in A_l$ between subnetworks to allow transition between modes of transportation. For lack of better data we add such links at public transit stations and points of interests. (Note that taxi stand data was not available to us, hence, we assume that transfers to and from a taxi can occur at any public transit station or point of interest.) Conceptionally, we add POIs as a separate subnetwork (with empty arc set), which we link to the walking network associating each POI with its nearest road vertex. Note that, typically, the road subnetworks are much larger in number of vertices and arcs, while the challenge in the public transit subnetworks lies in the time-dependency.

On such a network, quickest routes can be obtained by applying a variant [16, 28] of Dijkstra's algorithm [15]. Given a source node s, a target node t and a time of departure τ at the source, it computes the earliest arrival time at target t. For our purposes, though, we require to know quickest routes for all departure times of the day. This problem is known as *profile or range search* in the literature. It can be solved by label-correcting (LC) variants of Dijkstra's algorithm [8, 11, 24] that maintain as vertex labels travel time functions instead of scalar distances. For purely public transit networks label-setting approaches are known to perform better [9, 10, 14]. They exploit that for public transit networks travel time functions have special structure (see Figure 14), defined by departing connections. By easy transformation these functions can also be viewed as Pareto-sets of (departure time, arrival time) tuples. A route is Pareto-dominated iff it departs earlier but arrives later than another route. Under this perspective, LC [8] keeps Pareto-sets as vertex labels, processing the whole set at once when scanning outgoing arcs of a vertex. Hence, vertex rescans are expensive. Better query times can be achieved by (still) maintaining Pareto-sets as vertex labels (in an ordered fashion) but processing single elements at a time. In principle, this family

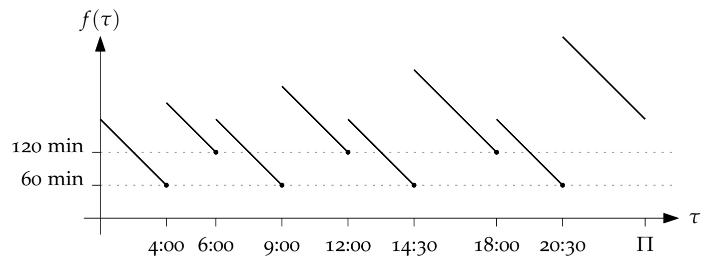


Figure 14: A time-dependent rail edge function representing seven departures: four fast trains with a travel-time of 60 minutes to the next stop, three slow trains with a travel-time of 120 minutes.

of algorithms [9, 10, 14] runs earliest arrival search for each departing train at the source stop. However, all these searches are cleverly interweaved to enable early detection of Pareto-inefficient solutions.

For multimodal networks such as ours, travel time functions look slightly different (see Figure 15), yet, the same considerations apply. In [4] we have studied both a label-correcting and a label-setting extension of Dijkstra's algorithm for multimodal profile queries. The first follows the algorithmic framework described in [11] but uses a specialized implementation that exploits the specific form of multimodal travel time functions. The latter is a label-setting extension that uses sets of labels of the form (departure time, travel time). When initialized on a road vertex, departure time will be undefined and travel time set to zero (corresponding to a constant-zero travel time function). Paths that start in the road network will hence have cost (undefined, path length in seconds). When such a path meets the the public transit network, its corresponding label is transformed: for each departure at the public transit vertex a label of form (departure time - path length, path length) is generated. All vertex labels are sorted by departure time since it enables Pareto-domination in linear time. The queue keeps vertex ids (not labels), with travel time as key. For each vertex, a single representative label is maintained—the one of lowest travel time that is still active: labels that were processed once are marked inactive and will never be reactivated.

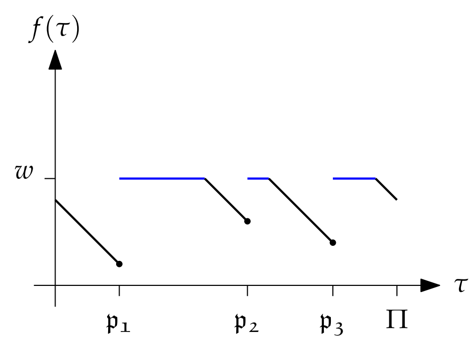


Figure 15: A mixed travel time function observing three characteristics: 1. Points in time $\mathfrak{p}_i$ of optimal journey departures. 2. Segments of slope -1, where waiting on the next departure results in optimal travel time. 3. Segments of slope 0, where, e. g., the walking duration w is an upper bound on the travel time (i. e., waiting on the next train departure does not pay off, e. g., in case of bad connectivity such as at nights).

Interestingly, experimental evaluation in [4] has found the label-correcting multimodal profile search to be slightly faster. More importantly, it showed that the bottleneck of both multimodal profile search algorithms is processing the road network.

4.4.2 Acceleration

In order to fasten the computation of the time-dependent distance matrix between points of interest we apply a quick preprocessing of the road networks (which make up the largest part of the multimodal network). The general approach has been described in [12], in the following we only give a short overview and then state the adaptations for our scenario.

We exploit the fact that modal transfers are restricted to a small set of vertices of each subnetwork. We therefore use preprocessing to compute a smaller *core graph* [43] that preserves distances between such transfer vertices $K \subset V$. More precisely, we start from the original graph and iteratively *contract* [25] each vertex in $V \setminus K$ in the order given by a rank function r. Each contraction step (temporarily) removes a vertex and adds shortcuts between its uncontracted neighbors to maintain shortest path distances (if necessary). It is usually advantageous to first contract vertices with relatively small degrees that are evenly distributed across the network [25]. We stop contraction when the average degree in the core graph reaches some threshold [12]. In practice, we only apply this preprocessing to the road networks.

To enable the computation of the multimodal time-dependent distance matrix, we, additionally, add all (road) vertices linked to POIs to the set K, keeping them in core. Then, for each POI, we can run a one-to-all multimodal profile query (as described in the previous subsection) from the associated POI vertex *restricted to the core* in order to obtain all multimodal time-dependent travel time distances.

4.4.3 Results

We implemented above algorithms in C++ and compiled with `g++` 4.7.1 (64 bits, flag `-O3`), running experiments on a single core of a 4x 12-core AMD Opteron-6172 machine, clocked at 2.1 GHz. .

The multimodal network of the Athens instance consists of public transit, walking, taxi, and points of interests (see Deliverable D3.2 for details). The public transit network has 7 778 stops, 570 routes, 26 192 trips, 1 003 188 daily departure events; in the time-dependent route model graph [42] this results in 29 055 vertices and 63 424 arcs. The walking network consists of 287 003 vertices and 685 850 arcs. The taxi network of 219 615 vertices and 472 591. Points of interests were obtained as described in Deliverable D3.2, however, the size of that data set has grown to 557 POIs total.

Preprocessing this network takes 162 seconds, after which 40 283 vertices remain in the core network. We run 557 multimodal one-to-all profile (24h range) queries on this core, and acquire a multimodal distance matrix with 71 145 759 entries total. Computing these distances takes approximately 105 minutes. The complete data set can be found at `http://i11www.iti.uni-karlsruhe.de/ecompass/wp3/benchmarks/`. Each of the 557 × 557 POI combinations has on average about 229 distinct walking/public transportation journeys throughout the day. The average walking/public transportation travel time is about 26 minutes, the maximal travel time is about 3:19 hours (of walking). In addition, we compute the travel time of a single, continuous taxi-ride per POI combination. The average travel time using the taxi is about 6 minutes, the maximal travel time is about 24 minutes.

4.4.4 Outlook

We have demonstrated a practical approach to precomputing multimodal time-dependent travel time distance matrices for applications such as the TTDP. However, at 105 minutes processing time we are far from being able to integrate real-time transit delay information (to be supplied by the public transit operator). Nor could we integrate road traffic information. While for walking this might be a non-issue, for taxi this could certainly make a difference. From a algorithmic point of view it could be worthwhile to look into adaptations of approaches described in [10, 14].

4.5 Experimental Results

4.5.1 Test Instances

While many different datasets exist for testing (T)OP(TW) problems, this is not the case for their time-dependent counterparts. To some extent, this is because only a limited body of literature focuses on the time dependent variants of OP; most importantly though, it is due to the difficulty in producing realistic synthesized multimodal timetabled data (respecting the FIFO property and the triangular inequality, among others). Hence, relevant algorithmic solutions should unavoidably be tested upon real transit network data (for instance, Garcia et al. [21] used timetabled data of the Sab Sebastian bus network, provided by the local transportation authority), to validate their solutions. Fortunately, the advent of the GTFS [1] (General Transit Feed Specification) standard, used by major transportation authorities worldwide to describe and publish their timetabled data, has made access to such data easier than before.

In our experiments, we have used the GTFS data of the transit network deployed on the metropolitan area of Athens, Greece, provided by the OASA (Athens Urban Transport Organization). The network comprises 3 subway lines, 3 tram lines and 287 bus lines with an overall of 7825 transit stops (see Section 4.4.3 for an analytical description of the multimodal network of the Athens). For our purposes, we require to know pairwise quickest routes between POIs, for all departure times of the day. Using the method for preprocessing multimodal travel time distances, as described in Section 4.4, we compute pairwise full (24h range) multimodal time-dependent travel time profiles. Namely, for each pair of POIs we compute S_{uv}, which contains all the non-dominated pairs $\mathrm{dep}_i^{uv}, \mathrm{trav}_i^{uv}, i = 1, 2, \ldots, |S_{uv}|$, 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 (in Athens $|S_{uv}| = 229$, on average). We also maintain the walking time among u and v, provided that this is shorter than using the transit network at any departure time within the day (otherwise, walking time is set to infinite). The overall shortest time dependent travel time information is pre-calculated and 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 (of size 3.5 GB in our implementation) ensures instant access to time dependent travel times, given a specified pair of POIs (u, v), upon receiving a user query.

We have segmented the Athens tourist area in 144 (12x12) square regions ($1\mathrm{km}^2$ each) to allow addressing the Generalized TTDP (see Section 4.3), i.e. allow associations of arbitrarily defined start/end locations to nearby area representatives (see Figure 16a). However, to validate our algorithmic solutions we have used a set of predefined start/end locations. In particular, we have used a set of 100 hotels scattered around the city, but mostly situated nearby POIs (see Figure 16b). Those serve as potential start/end locations, i.e. we assume that typical tourist routes start and end at the tourist's accommodation location.

The POIs dataset used in our experiments features 113 sites (museums and art galleries, archaeological sites, monuments & landmarks, streets & squares, neighborhoods, churches & religious heritage, nature) mostly situated around Athens downtown and Piraeus areas (see Figure 16a). The POIs have been compiled from various tourist portals [2] and web services offering open APIs [3]. 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). 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). The POIs have been grouped in $\lfloor \frac{N}{10} \rfloor = 11$ disjoint clusters.

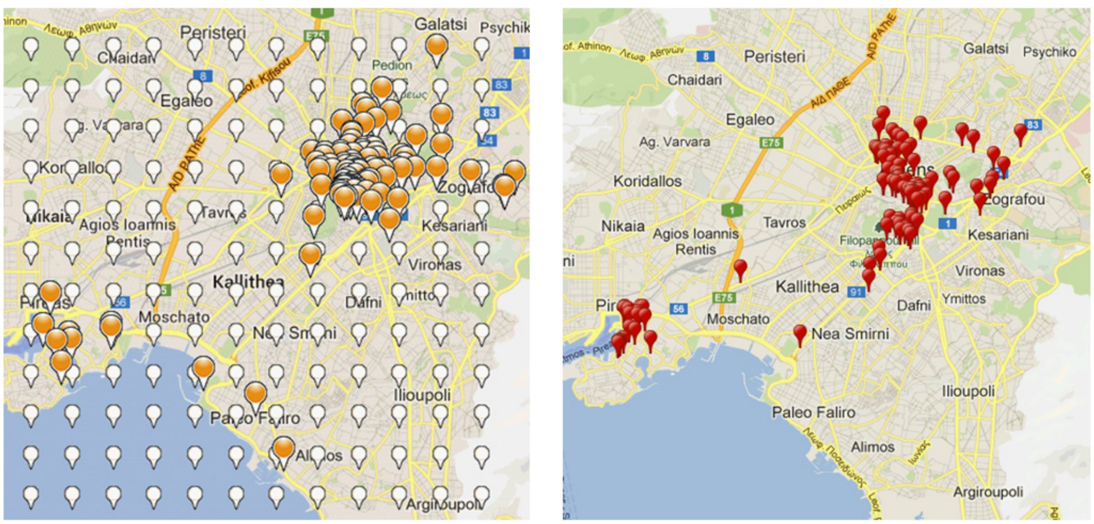
The above described POIs dataset has been used to create three different 'topologies' (referred to as 'topol1', 'topol2' and 'topol3' in the tables shown on the next subsection): the real POIs

[1] https://developers.google.com/transit/gtfs/reference
[2] http://www.tripadvisor.com/, http://index.pois.gr/
[3] https://developers.google.com/places/documentation/

coordinates have been maintained in all cases, however, their respective profits, visiting times and opening hours (i.e. time windows) have been 'shuffled', to ensure a fair validation of the evaluated algorithms removing any potential bias of a single topology.

Our algorithms have been tested using 100 different 'user preference' inputs, 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) a POI is disregarded on all routes with a 10% probability (this 'simulates' preferences provided by real visitors, such as no interest on religious sites, which enables the algorithms to disregard a subset of available POIs), and (b) each of the remaining POIs is removed from a specific route with a 10% probability (this caters for the possibility of unsuitable weather conditions; for instance, a TTDP solver should disqualify a visit to an open-air POI in a rainy day [4]). 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`



(a) (b)

Figure 16: (a) Area centers (white markers) and POIs locations (orange markers) in the Athens metropolitan area; (b) Hotels locations.

4.5.2 Results

We have implemented the following five algorithms: (a) TDCSCRoutes (see Section 4.2.2), (b) SlackCSCRoutes (see Section 4.2.3), (c) Time Dependent ILS (TDILS - in effect, this is an extension of the standard ILS TOPTW algorithm [50], wherein we take into account time dependent, rather than constant, travel times in the insertion of nodes into routes), (d) AvgCSCRoutes (see Section 4.2.4), and (e) Average ILS (AvgILS).

AvgILS refers to the average travel time approach proposed by Garcia et al. [21], wherein TDTOPTW is practically reduced to TOPTW and the standard ILS algorithm [50] 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

[4] Weather forecast information may be easily retrieved from freely available web services like Yahoo Weather (http://weather.yahoo.com/) and fed into the TTDP solver.

a visit to become infeasible, the latter is removed from the route and the remainder of the route is shifted forward. AvgCSCRoutes employs a similar repair step and then a 'gap filling' step (see steps No 3 and 4 in Section 4.2.4); the latter inserts new POIs into the routes, if feasible, thereby further improving the solution's quality.

All algorithms have been employed upon the test instances described in the previous subsection, deriving k daily personalized routes, $k = 1..4$, each for every day of stay at the destination. All routes start and end at the tourist's accommodation location (see Figure 17).

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.

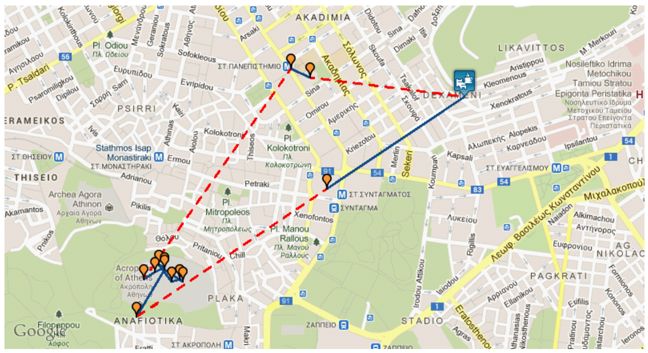


Figure 17: Illustration of a tourist route starting/ending at a hotel; solid lines indicate walking transfers while dashed lines indicate public transit transfers.

Our experimental results comprise two sets. In the first result set (see Section 4.5.2.1) we use the standard profit criterion, i.e. we consider as best-found solution the one with the highest aggregate profit. In the second result set (see Section 4.5.2.2) we use a 'walk motivation' criterion, i.e. we select the solution that maximizes profit$(2 + \frac{1}{\text{transit} + 1})$, where *profit* equals the aggregate profit over all routes and *transit* equals the overall transit transfers occurring along all routes. The latter criterion clearly favors the insertion of POIs within walking distance from their previous and next POIs.

When employing the walk motivation criterion, all algorithm implementations exercise an additional repair step: transit transfers are substituted by walking transfers wherever this does not impact the actual solution. This repair step is executed when reaching a local optimum for TD-CSCRoutes, SlackCSCRoutes and TDILS or when deriving the final solution for AvgCSCRoutes and AvgILS. In effect, this favors walking over transit transfers, given the reluctance of tourists in using public transit, especially when considering relatively short distances.

4.5.2.1. Results – Profit Criterion

Tables 11 – 15 illustrate the experimental results compiled for the five implemented algorithms when employing the standard profit criterion. The tables include results yield for the three topologies of the Athens dataset (see Section 4.5.1) and for 1-4 daily tourist routes. Table 16 offers a comparative view on the algorithms performance.

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 walking/transit transfers (e.g. for one route, that is the number of visited POIs plus one to return back to the accommodation); the overall number of public transit transfers (PT) over all routes; the percentage of the public transit transfers over the overall transfers (PT%); the aggregate time spent for waiting until starting a visit (i.e. from arrival until the opening time of a POI) plus the delays for embarking on public transit (in practice wait time is negligible due to the relatively wide and overlapped time windows); the average delay time for embarking on public transit. All the above results are averaged over all (= 100) the execution runs (hence, the decimal numbers for the number of visits and transfers). High quality solutions are those featuring high aggregate profit and relatively small number of transit transfers, derived in short execution time. It is noted that Table 16 averages results over the three topologies, while also normalizing 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 a general remark applied to all algorithms, the increase of the overall collected profit with the increase of the number of routes is not linear, since the average POI profits is higher when considering low numbers of routes (i.e. for short stays, the tourist only visits the not-miss POIs). The same applies to the number of visits, as it appears that shorter stays tend to favor visits to best profit-for-time POIs (hence, they include larger number or POIs per route) disregarding those associated with long visiting time. Furthermore, all algorithms perform remarkably well as far as the average delay for transit transfers is concerned (typically less than two minutes per transfer).

As expected, the algorithms working with average travel times (i.e. AvgCSCRoutes and AvgILS) execute considerably faster, since they disregard time dependency on the insertion decision, while also using smaller memory structures to hold travel time information (hence, required travel times are retrieved more efficiently). AvgILS executes slower than AvgCSCRoutes as it explores a larger search space on each POI insertion. Interestingly, AvgCSCRoutes and AvgILS are competitive in terms of profit (although they perform worse than TDILS and TDCSCRoutes), 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 main execution (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 (e.g. frequent services in peak hours and infrequent services in off-peak hours) or even when considering tourist visits in off-peak hours (e.g. afternoon to night time budgets). In such scenarios, using the average travel time would not serve as a good approximation.

TDILS performs marginally better than TDCSCRoutes and SlackCSCRoutes with respect to the overall profit (achieving a performance gap up to 0.35% from TDCSCRoutes and up to 2,43% from SlackCSCRoutes). This is mainly because both TDCSCRoutes and SlackCSCRoutes, when deciding on the best candidate node to insert between a pair of POIs, they are typically restricted in considering exclusively POIs grouped within the same clusters, thereby compromising the quality of their solutions.

On the other hand, SlackCSCRoutes performs marginally better than TDILS with respect to the number of transits, while also achieving higher number of POI visits (intuitively, in order to achieve high slack variable values, the algorithm favors insertions of nodes reachable via walking).

Notably, TDILS requires significantly longer execution time compared to TDCSCRoutes and SlackCSCRoutes, mainly due to exploring a much larger search space in node insertions, while also executing larger number of iterations in search of improved solutions.

Table 11: Results for TDCSCRoutes

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1106.40	72.15	14.50	15.50	2.84	18.32	2.98	1.05
2	1772.25	189.05	24.48	26.48	4.66	17.60	5.35	1.15
3	2268.40	369.17	32.49	35.49	6.54	18.43	8.17	1.25
4	2670.15	586.82	39.88	43.88	8.20	18.69	11.71	1.43
topol2								
1	861.15	58.38	11.86	12.86	3.67	28.54	3.80	1.04
2	1520.80	161.47	21.81	23.81	6.17	25.91	7.59	1.23
3	2029.85	321.97	29.75	32.75	9.24	28.21	13.13	1.42
4	2475.10	589.65	37.30	41.30	12.02	29.10	17.19	1.43
topol3								
1	903.15	60.12	12.26	13.26	2.67	20.14	2.82	1.06
2	1553.25	166.25	21.52	23.52	5.77	24.53	7.82	1.36
3	2067.60	322.31	29.16	32.16	8.38	26.06	14.06	1.68
4	2515.35	537.34	36.71	40.71	10.66	26.19	18.84	1.77

Table 12: Results for SlackCSCRoutes

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1107.35	80.00	14.67	15.67	2.76	17.61	3.23	1.17
2	1756.65	201.62	24.78	26.78	4.40	16.43	5.96	1.35
3	2220.35	341.57	33.29	36.29	6.00	16.53	10.43	1.74
4	2618.80	511.73	41.49	45.49	7.44	16.36	14.12	1.90
topol2								
1	860.35	66.38	12.27	13.27	3.40	25.62	4.32	1.27
2	1512.05	176.66	22.42	24.42	6.07	24.86	9.40	1.55
3	2001.15	319.88	30.92	33.92	8.37	24.68	14.02	1.68
4	2429.40	478.68	38.97	42.97	11.21	26.09	19.94	1.78
topol3								
1	902.75	65.73	12.64	13.64	2.37	17.38	2.83	1.19
2	1538.05	177.23	21.93	23.93	5.37	22.44	9.68	1.80
3	2026.30	318.55	29.92	32.92	7.49	22.75	14.82	1.98
4	2450.35	502.50	37.78	41.78	10.32	24.70	20.32	1.97

Table 13: Results for TDILS

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1108.15	125.29	14.54	15.54	2.88	18.53	3.21	1.11
2	1777.30	409.70	24.48	26.48	4.45	16.81	5.12	1.15
3	2276.30	736.10	32.60	35.60	6.17	17.33	8.17	1.32
4	2679.05	976.22	39.95	43.95	7.81	17.77	10.77	1.38
topol2								
1	858.95	95.83	11.68	12.68	3.71	29.26	3.84	1.04
2	1524.90	364.06	21.94	23.94	6.06	25.31	7.37	1.22
3	2038.00	690.11	29.78	32.78	8.84	26.97	11.86	1.34
4	2486.45	975.82	37.21	41.21	11.40	27.66	15.74	1.38
topol3								
1	902.90	96.04	12.18	13.18	2.43	18.44	2.21	0.91
2	1554.55	336.56	21.47	23.47	5.64	24.03	7.40	1.31
3	2073.90	667.82	29.23	32.23	7.90	24.51	12.29	1.56
4	2519.90	927.03	36.53	40.53	10.06	24.82	16.25	1.62

Table 14: Results for AvgCSCRoutes

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1094.10	43.43	14.32	15.32	2.57	16.78	3.60	1.40
2	1757.40	109.31	24.45	26.45	4.30	16.26	6.86	1.60
3	2250.20	197.48	32.41	35.41	5.82	16.44	10.83	1.86
4	2661.70	314.77	39.78	43.78	7.17	16.38	14.17	1.98
topol2								
1	841.40	36.29	11.65	12.65	3.33	26.32	4.06	1.22
2	1500.95	92.91	21.60	23.60	5.57	23.60	8.60	1.54
3	2010.45	177.22	29.85	32.85	7.35	22.37	12.61	1.72
4	2452.35	290.35	37.29	41.29	10.34	25.04	22.51	2.18
topol3								
1	888.60	39.02	12.19	13.19	2.25	17.06	2.66	1.18
2	1530.65	100.42	21.51	23.51	4.76	20.25	8.85	1.86
3	2044.90	180.14	29.41	32.41	6.22	19.19	12.74	2.05
4	2487.65	291.94	36.94	40.94	8.03	19.61	16.62	2.07

Table 15: Results for AvgILS

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1079.85	64.80	13.85	14.85	2.50	16.84	4.81	1.92
2	1748.05	213.70	23.90	25.90	4.17	16.10	7.79	1.87
3	2236.05	406.79	31.86	34.86	5.75	16.49	12.25	2.13
4	2645.20	505.49	39.07	43.07	7.21	16.74	15.62	2.17
topol2								
1	823.85	50.11	11.07	12.07	3.08	25.52	4.99	1.62
2	1478.35	183.83	21.08	23.08	5.41	23.44	9.52	1.76
3	1988.85	336.31	29.10	32.10	7.24	22.55	15.15	2.09
4	2424.05	523.93	36.30	40.30	10.22	25.36	24.06	2.35
topol3								
1	871.70	54.02	11.78	12.78	2.30	18.00	4.01	1.74
2	1505.60	195.03	20.79	22.79	4.53	19.88	9.62	2.12
3	2017.00	357.91	28.49	31.49	6.04	19.18	13.85	2.29
4	2454.05	491.05	35.81	39.81	8.01	20.12	19.52	2.44

Table 16: Comparative view of results compiled for the tests employing the standard profit criterion

#Routes		Profit	Time	Visits	PT
1	TDCSCRoutes	**100**	60.11	97.57	100
	SlackCSCRoutes	99.99	66.88	**100**	92.92
	TDILS	99.98	100	97.02	98.26
	AvgCSCRoutes	98.38	**37.44**	96.41	88.78
	AvgILS	96.68	53.26	92.72	**85.84**
2	TDCSCRoutes	99.78	46.54	98.09	100
	SlackCSCRoutes	98.97	50.03	**100**	95.42
	TDILS	**100**	100	98.21	97.29
	AvgCSCRoutes	98.61	**27.26**	97.73	88.13
	AvgILS	97.43	53.37	95.14	**85**
3	TDCSCRoutes	99.65	48.4	97.1	100
	SlackCSCRoutes	97.8	46.8	**100**	90.48
	TDILS	**100**	100	97.32	94.83
	AvgCSCRoutes	98.71	**26.5**	97.39	80.26
	AvgILS	97.71	52.58	95.03	**78.77**
4	TDCSCRoutes	99.68	59.53	96.32	100
	SlackCSCRoutes	97.57	51.85	**100**	93.81
	TDILS	**100**	100	96.15	94.79
	AvgCSCRoutes	98.91	**31.16**	96.42	82.71
	AvgILS	97.89	52.81	94.03	**82.38**

4.5.2.2. Results – Walk motivation Criterion

Tables 17 – 21 illustrate the experimental results compiled for the five implemented algorithms when employing the 'walk motivation' instead of the profit criterion, while Table 22 offers a comparative view on the algorithms performance. The results indicate a clear tradeoff between profit and number of transit transfers. In particular, since profit is not the sole criterion used for picking the best solution, the overall profit is reduced compared to the results discussed in the previous subsection. On the other hand, the incorporation of the occurring transit transfers into the criterion for finding best solutions has considerably reduced the overall number of transit transfers along the derived routes (typically, each route features one transit transfer less than in the previous result sets). Notably, the profit values associated with AvgCSCRoutes and AvgILS are identical with those shown in Tables 14 – 15, as both these algorithms lack transit-related information in their main execution phase, hence, they are unable to effectively employ the 'walk motivation' criterion.

TDILS maintains its prevalence over TDCSCRoutes and SlackCSCRoutes with respect to solutions quality (i.e. overall profit). However, SlackCSCRoutes achieves a significant performance gap in terms of the number of transit transfers (half as many as in TDILS), reducing accordingly the total delay time experienced along the routes. This is due to the objective of SlackCSCRoutes to maximize the available time between successive visits for selecting slower transportation modes, such as walking. Thus, when motivating walks, SlackCSCRoutes considerably reduces the transit transfers in favor of walking. Furthermore, SlackCSCRoutes appears to accommodate the highest number of POI visits among all tested algorithms.

Table 17: Results for TDCSCRoutes

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1094.05	72.13	14.38	15.38	1.34	8.71	1.61	1.20
2	1755.75	191.94	24.32	26.32	2.30	8.74	2.78	1.21
3	2250.65	380.42	32.28	35.28	3.61	10.23	5.01	1.39
4	2639.80	607.67	38.96	42.96	4.47	10.41	6.51	1.46
topol2								
1	838.05	59.24	11.48	12.48	1.43	11.46	1.63	1.14
2	1489.70	161.61	21.17	23.17	3.53	15.24	4.29	1.22
3	2022.55	318.42	29.61	32.61	6.80	20.85	9.18	1.35
4	2473.65	582.77	37.29	41.29	9.75	23.61	13.40	1.37
topol3								
1	889.85	61.83	12.00	13.00	0.89	6.85	0.97	1.09
2	1523.10	167.90	21.07	23.07	2.98	12.92	4.08	1.37
3	2044.55	328.42	29.04	32.04	5.17	16.14	9.01	1.74
4	2504.75	549.69	36.64	40.64	7.89	19.41	13.62	1.73

Table 18: Results for SlackCSCRoutes

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1093.10	82.09	14.58	15.58	1.02	6.55	1.53	1.50
2	1735.50	205.12	24.79	26.79	1.60	5.97	2.83	1.77
3	2199.30	353.82	33.15	36.15	2.46	6.80	5.36	2.18
4	2587.70	519.09	41.17	45.17	3.14	6.95	7.64	2.43
topol2								
1	838.80	67.25	11.94	12.94	1.01	7.81	1.89	1.87
2	1478.40	179.95	22.00	24.00	2.45	10.21	4.05	1.65
3	1965.40	319.92	30.71	33.71	3.95	11.72	7.57	1.92
4	2390.90	501.56	38.95	42.95	5.91	13.76	10.97	1.86
topol3								
1	894.95	66.64	12.49	13.49	0.67	4.97	0.97	1.45
2	1512.60	181.46	21.89	23.89	2.02	8.46	4.25	2.10
3	1990.10	333.63	29.88	32.88	3.15	9.58	7.59	2.41
4	2408.90	500.68	37.66	41.66	5.14	12.34	11.41	2.22

Table 19: Results for TDILS

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1102.10	128.50	14.39	15.39	2.21	14.36	2.43	1.10
2	1776.05	409.89	24.46	26.46	3.67	13.87	3.96	1.08
3	2276.30	735.78	32.60	35.60	5.17	14.52	6.45	1.25
4	2678.25	976.06	39.94	43.94	6.11	13.91	8.49	1.39
topol2								
1	855.65	94.82	11.64	12.64	2.83	22.39	2.65	0.94
2	1522.55	354.15	21.89	23.89	5.02	21.01	5.75	1.15
3	2038.00	673.25	29.78	32.78	7.06	21.54	8.74	1.24
4	2486.45	951.63	37.21	41.21	9.49	23.03	12.82	1.35
topol3								
1	899.30	100.58	12.08	13.08	1.61	12.31	1.34	0.83
2	1552.30	340.59	21.43	23.43	4.38	18.69	5.53	1.26
3	2073.90	668.57	29.23	32.23	6.45	20.01	9.89	1.53
4	2519.90	927.04	36.53	40.53	8.27	20.40	13.59	1.64

Table 20: Results for AvgCSCRoutes

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1094.10	43.84	14.32	15.32	1.55	10.12	2.06	1.33
2	1757.40	109.45	24.45	26.45	2.90	10.96	4.95	1.71
3	2250.20	197.97	32.41	35.41	3.72	10.51	7.72	2.08
4	2661.70	315.75	39.78	43.78	4.81	10.99	10.75	2.23
topol2								
1	841.40	36.23	11.65	12.65	2.19	17.31	2.35	1.07
2	1500.95	92.86	21.60	23.60	3.66	15.51	4.90	1.34
3	2010.45	177.27	29.85	32.85	4.77	14.52	8.68	1.82
4	2452.35	290.84	37.29	41.29	6.74	16.32	15.93	2.36
topol3								
1	888.60	39.13	12.19	13.19	1.35	10.24	1.75	1.30
2	1530.65	101.13	21.51	23.51	3.14	13.36	6.68	2.13
3	2044.90	181.52	29.41	32.41	4.05	12.50	8.85	2.19
4	2487.65	294.53	36.94	40.94	5.70	13.92	13.42	2.35

Table 21: Results for AvgILS

# Routes	Profit	Time (ms)	Visits	Transfers	PT	PT (%)	Wait + Delay (min)	Avg Delay
topol1								
1	1079.85	64.85	13.85	14.85	1.41	9.49	2.88	2.04
2	1748.05	214.52	23.90	25.90	2.43	9.38	4.96	2.04
3	2236.05	406.50	31.86	34.86	3.28	9.41	7.96	2.43
4	2645.20	505.60	39.07	43.07	4.21	9.77	11.18	2.66
topol2								
1	823.85	50.09	11.07	12.07	1.43	11.85	2.24	1.57
2	1478.35	184.01	21.08	23.08	2.90	12.56	5.37	1.85
3	1988.85	336.27	29.10	32.10	3.84	11.96	9.38	2.44
4	2424.05	523.75	36.30	40.30	5.82	14.44	16.54	2.84
topol3								
1	871.70	54.32	11.78	12.78	1.10	8.61	2.35	2.14
2	1505.60	195.32	20.79	22.79	2.42	10.62	5.95	2.46
3	2017.00	358.51	28.49	31.49	3.48	11.05	9.62	2.76
4	2454.05	492.05	35.81	39.81	4.96	12.46	14.05	2.83

Table 22: Comparative view of results compiled for the tests employing the ‘walk motivation’ criterion

#Routes		Profit	Time	Visits	Public Transport
1	TDCSCRoutes	98.77	59.65	97.05	55.04
	SlackCSCRoutes	98.94	66.68	**100**	**40.6**
	TDILS	**100**	100	97.69	100
	AvgCSCRoutes	98.85	**36.8**	97.82	76.54
	AvgILS	97.14	52.26	94.08	59.25
2	TDCSCRoutes	98.3	47.21	96.91	67.41
	SlackCSCRoutes	97.44	51.29	**100**	**46.44**
	TDILS	**100**	100	98.69	100
	AvgCSCRoutes	98.72	**27.47**	98.37	74.22
	AvgILS	97.55	53.76	95.76	59.3
3	TDCSCRoutes	98.9	49.44	97	83.4
	SlackCSCRoutes	96.35	48.49	**100**	**51.18**
	TDILS	**100**	100	97.73	100
	AvgCSCRoutes	98.71	**26.8**	97.79	67.13
	AvgILS	97.71	53.01	95.42	56.75
4	TDCSCRoutes	99.14	60.96	95.85	92.63
	SlackCSCRoutes	96.13	53.29	**100**	**59.45**
	TDILS	**100**	100	96.52	100
	AvgCSCRoutes	98.92	**31.57**	96.8	72.27
	AvgILS	97.9	53.29	94.4	62.8

4.6 Conclusions and Future Work

We introduced TDCSCRatio and SlackCSCRoutes, two novel approaches to the TTDP, which allow modeling multimodal transfers among POIs. To the best of our knowledge, these are the only TDTOPTW solvers not based in the unrealistic assumption of periodic transit services. 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.

Our experimental results demonstrate that the use of the 'walk motivation' criterion satisfies best the requirements of most tourists (i.e. it minimizes transit transfers), at the expense of marginally lower solutions quality.

With respect to the overall collected profit, TDILS has been shown to perform better; however, it appears to be suitable only when considering small to medium-scale datasets, as it requires longer execution phases. In practical applications, comprising large datasets, AvgCSCRoutes could be the most suitable choice as it efficiently derives solutions of reasonably good quality. Nevertheless, its suitability largely depends on the high frequency of public transit services, so that average travel time would be a good approximation.

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 (respecting the FIFO property and the triangular inequality, among others) to serve as additional test benchmarks.

Our future work will also focus on variants of TOPTW to tackle more realistic TTDPs. For instance, tourists typically require relaxing and having breaks (e.g. for coffee and meal) in between of visits to POIs. Such breaks are typically specific in number, while respective recommendations may be subject to strict time window (e.g. meal should be scheduled around noon) and budget constraints. Further, we plan to incorporate max-n type [45] restrictions to constrain the selection of POIs by allowing users to state a maximum number of certain types of POIs, per day or for the whole trip (e.g., maximum two museum visits on the first day). Likewise, mandatory visits (i.e. tours including at least one visit to a POI of certain type, such as a visit to a church) could also be asked for. Focused adjustments and refinements of our algorithms should be able to provide such features.

References

[1] R. A. Abbaspour and F. Samadzadegan. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems and Applications*, 38:12439–12452, 2011.

[2] C. Archetti, A. Hertz, and M. Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13:49–76, 2007.

[3] M. Bagirov. Modified global k-means algorithm for minimum sum-of-squares clustering problems. *Pattern Recognition*, 41(10):3192 – 3199, 2008.

[4] Andreas Bauer. Multimodal Profile Queries. Technical Report no. 020, eCOMPASS Project, June 2013.

[5] T. Beer, M. Fuchs, W. Höpken, J. Rasinger, and H. Werthner. Caips: A context-aware information push service in tourism. In *Proceedings of the International Conference on Information and Communication Technologies in Tourism*, pages 129–140. Springer Vienna, 2007.

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

[7] 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.

[8] Brian C. Dean. Continuous-Time Dynamic Shortest Path Algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[9] Daniel Delling, Bastian Katz, and Thomas Pajor. Parallel computation of best connections in public transportation networks. Technical Report no. 013, eCOMPASS Project, July 2012.

[10] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

[11] Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

[12] Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multi-Modal Route Planning. Technical Report no. 006, eCOMPASS Project, April 2012.

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

[14] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. Technical Report no. 021, eCOMPASS Project, June 2013.

[15] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[16] Stuart E. Dreyfus. An Appraisal of Some Shortest-Path Algorithms. *Operations Research*, 17(3):395–412, 1969.

[17] F. V. Fomin and A. Lingas. Approximation algorithms for time-dependent orienteering. *Information Processing Letters*, 83(2):57 – 62, 2002.

[18] 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.

[19] A. Garcia, O. Arbelaitz, P. Vansteenwegen, W. Souffriau, and M. Linaza. Hybrid approach for the public transportation time dependent orienteering problem with time windows. In *Proceedings of the 5th International Conference on Hybrid Artificial Intelligence Systems (HAIS'2010)*, pages 151–158. 2010.

[20] A. Garcia, M. Linaza, O. Arbelaitz, and P. Vansteenwegen. Intelligent routing system for a personalised electronic tourist guide. In W. Hpken, U. Gretzel, and R. Law, editors, *Information and Communication Technologies in Tourism 2009*, pages 185–197. Springer Vienna, 2009.

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

[22] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and Y. Tasoulas. A survey on algorithmic approaches for solving tourist trip design problems. http://www.ecompass-project.eu/sites/default/files/ECOMPASS-TR-010.pdf, October 2012. Technical Report of European Commission FP7 Project 'eCO-friendly urban Multi-modal route PlAnning Services for mobile uSers (eCOMPASS)'.

[23] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and Y. Tasoulas. Cluster-based heuristics for the team orienteering problem with time windows. In *Proceedings of 12th International Symposium on Experimental Algorithms (SEA'13)*, pages 390–401, 2013.

[24] Robert Geisberger. Contraction of Timetable Networks with Realistic Transfers. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer, May 2010.

[25] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

[26] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[27] S. Gössling, P. Peeters, J.P. Ceron, G. Dubois, T. Patterson, and R.B. Richardson. The eco-efficiency of tourism. *Ecological Economics*, 54(4):417 – 434, 2005.

[28] David E. Kaufman and Robert L. Smith. Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

[29] 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.

[30] N. Labadi, J. Melechovský, and R. Calvo. An effective hybrid evolutionary local search for orienteering and team orienteering problems with time windows. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Gnter Rudolph, editors, *Parallel Problem Solving from Nature PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 219–228. Springer Berlin / Heidelberg, 2010.

[31] 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.

[32] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193 – 207, 1990.

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

[34] J. Li. Model and algorithm for time-dependent team orienteering problem. In S. Lin and X. Huang, editors, *Advanced Research on Computer Education, Simulation and Modeling*, volume 175 of *Communications in Computer and Information Science*, pages 1–7. Springer Berlin Heidelberg, 2011.

[35] J. Li, Q. Wu, X. Li, and D. Zhu. Study on the time-dependent orienteering problem. In *Proceedings of the 2010 International Conference on E-Product E-Service and E-Entertainment (ICEEE'2010)*, pages 1–4, nov. 2010.

[36] 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.

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

[38] 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.

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

[40] 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.

[41] S. Page. *Transport and tourism*. Pearson Prentice Hall, 2nd edition, 1999.

[42] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

[43] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. *ACM Journal of Experimental Algorithmics*, 5(12):1–23, 2000.

[44] M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35:254–265, 1987.

[45] W. Souffriau and P. Vansteenwegen. Tourist trip planning functionalities: State of the art and future. In *Proceedings of the 10th International Conference on Current Trends in Web Engineering (ICWE'10)*, pages 474–485. 2010.

[46] F. C. R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 1999.

[47] 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.

[48] P. Vansteenwegen. *Planning in Tourism and Public Transportation - Attraction Selection by Means of a Personalised Electronic Tourist Guide and Train Transfer Scheduling*. PhD thesis, Katholieke Universiteit Leuven, 2008.

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

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

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

[52] P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: An or opportunity. *Operational Research Insight*, 20(3):21–27, 2007.

[53] B. Zenker and B. Ludwig. Rose: assisting pedestrians to find preferred events and comfortable public transport connections. In *Proceedings of the 6th International Conference on Mobile Technology, Application, Systems*, Mobility '09, pages 16:1–16:5, 2009.

Appendix A Analytical Results for TOPTW Algorithms

Table 23: Results for Solomon instances for 1 tour

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
c101	10	320	7	104	310	8	94	300	5	92
c102	10	360	6	95	360	6	128	360	6	111
c103	10	390	8	101	390	8	201	380	8	159
c104	10	400	9	128	420	7	217	410	7	205
c105	10	340	9	97	340	11	122	330	7	99
c106	10	340	8	101	340	9	130	330	6	124
c107	10	360	8	104	360	8	142	360	6	113
c108	10	370	8	124	370	8	158	360	6	129
c109	10	380	8	111	380	6	172	380	6	144
c201	10	840	16	553	860	14	370	840	10	161
c202	10	910	14	780	910	12	469	890	9	146
c203	10	940	19	726	940	14	647	900	10	169
c204	10	950	17	549	960	11	843	970	10	235
c205	10	900	13	406	900	12	476	890	10	167
c206	10	910	15	419	920	12	513	900	10	190
c207	10	910	17	623	930	12	570	920	10	203
c208	10	930	14	463	930	13	618	920	10	201
r101	10	182	6	61	183	7	64	180	5	70
r102	10	286	6	111	286	6	122	282	5	104
r103	10	286	7	101	291	6	166	289	6	127
r104	10	297	6	117	301	4	167	303	5	151
r105	10	247	5	135	247	5	91	238	3	85
r106	10	293	6	108	293	6	128	279	5	116
r107	10	288	7	100	294	5	162	289	6	119
r108	10	297	5	170	308	5	187	303	5	146
r109	10	276	7	124	276	7	112	259	3	95
r110	10	281	4	144	281	4	133	281	4	109
r111	10	295	6	129	295	6	160	297	4	119
r112	10	295	6	114	295	6	185	285	3	132
r201	10	788	28	709	786	23	491	476	10	115
r202	10	880	27	965	891	21	654	788	10	161
r203	10	980	22	1781	983	22	975	914	10	231
r204	10	1073	23	909	1057	16	1380	1048	10	291
r205	10	931	28	1452	905	28	815	644	10	144
r206	10	996	21	701	996	22	915	849	10	206
r207	10	1038	21	789	1059	20	1101	917	10	259
r208	10	1069	17	1524	1083	17	1437	1061	10	297
r209	10	926	22	706	920	23	963	717	10	221
r210	10	958	24	1132	970	21	976	813	10	200
r211	10	1023	24	728	1025	17	1216	865	10	274
rc101	10	219	4	93	219	4	78	219	4	79
rc102	10	259	5	117	259	5	98	266	4	81
rc103	10	265	6	100	263	4	115	266	4	95
rc104	10	297	5	83	301	4	133	301	4	110
rc105	10	221	4	116	244	4	93	241	4	86
rc106	10	239	5	124	250	4	95	250	4	86
rc107	10	274	5	124	276	5	110	261	4	102
rc108	10	288	5	104	288	5	125	274	4	111
rc201	10	780	19	583	777	19	384	646	10	121
rc202	10	882	19	762	924	14	543	864	10	165
rc203	10	960	13	761	956	18	750	901	10	173
rc204	10	1117	14	852	1108	11	1100	1121	10	279
rc205	10	840	17	564	845	15	435	685	10	138
rc206	10	860	19	541	869	18	568	751	10	140
rc207	10	926	17	896	925	15	771	801	10	205
rc208	10	1037	17	1226	1026	16	884	971	10	254

Table 24: Results for Cordeau et al. instances for 1 tour

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
pr01	4	304	8	109	304	8	77	236	4	44
pr02	9	385	5	371	389	8	211	374	4	112
pr03	14	384	9	343	393	11	298	349	6	161
pr04	19	447	18	879	464	9	473	425	7	236
pr05	24	576	14	1221	552	12	790	446	7	353
pr06	28	538	13	1223	554	16	879	472	9	516
pr07	7	291	6	120	291	5	132	291	5	79
pr08	14	463	7	615	446	5	314	397	5	164
pr09	21	461	10	685	468	8	497	442	5	303
pr10	28	539	13	1235	528	14	933	492	7	494
pr11	4	330	5	136	340	8	97	321	3	86
pr12	9	431	5	355	434	5	293	408	3	141
pr13	14	450	7	476	447	8	426	421	6	237
pr14	19	482	9	749	505	6	687	465	8	312
pr15	24	638	17	1531	636	18	1171	594	9	409
pr16	28	559	10	4195	577	11	1290	525	6	591
pr17	7	346	9	247	349	7	194	331	4	93
pr18	14	479	10	821	523	12	401	408	6	188
pr19	21	499	9	1131	510	8	738	487	8	424
pr20	28	570	16	1844	595	14	1264	522	12	569

Table 25: Results for Solomon instances for 2 tours

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
c101	10	590	15	278	590	12	195	550	8	141
c102	10	650	11	410	650	14	255	650	11	220
c103	10	700	14	305	710	13	407	700	12	310
c104	10	750	14	534	750	14	506	740	12	408
c105	10	640	16	288	640	16	218	600	12	196
c106	10	620	17	282	620	17	263	600	12	241
c107	10	670	13	304	670	13	259	620	11	184
c108	10	670	16	299	670	16	311	640	9	197
c109	10	710	10	378	720	10	398	690	9	289
c201	10	1400	24	1115	1420	21	838	1430	15	375
c202	10	1430	29	873	1430	23	1164	1440	18	385
c203	10	1430	28	848	1440	25	1365	1440	15	399
c204	10	1460	28	1080	1450	26	1491	1460	15	466
c205	10	1450	18	1791	1450	18	1098	1440	17	487
c206	10	1440	17	923	1470	15	1304	1470	14	450
c207	10	1450	22	1078	1470	16	1184	1470	14	504
c208	10	1460	19	1178	1480	17	1478	1460	16	540
r101	10	330	12	180	343	10	124	325	7	107
r102	10	508	11	290	501	12	255	501	9	174
r103	10	513	13	292	514	12	327	504	8	258
r104	10	539	10	346	543	10	397	529	10	287
r105	10	430	10	252	442	10	171	422	7	153
r106	10	529	12	411	524	12	256	505	10	199
r107	10	529	11	332	524	10	310	523	10	252
r108	10	549	11	345	556	9	386	552	9	308
r109	10	498	11	376	506	12	258	480	7	173
r110	10	515	9	455	508	8	276	506	7	255
r111	10	535	10	605	538	10	326	538	10	217
r112	10	515	10	523	538	9	423	531	7	312
r201	10	1231	54	838	1212	46	1299	864	19	269
r202	10	1270	47	955	1302	44	1285	1115	20	447
r203	10	1377	46	726	1372	40	1304	1243	20	560
r204	10	1440	38	565	1438	32	1453	1364	19	637
r205	10	1338	40	913	1333	39	1390	1115	20	413
r206	10	1401	44	667	1406	41	1672	1294	20	458
r207	10	1428	48	588	1434	44	1574	1378	20	758
r208	10	1458	43	456	1458	36	1667	1419	20	597
r209	10	1345	48	933	1370	40	1702	1187	19	429
r210	10	1365	42	915	1383	41	1438	1281	20	465
r211	10	1422	37	656	1438	35	2073	1316	18	552
rc101	10	427	7	506	427	7	150	419	8	130
rc102	10	494	7	355	488	8	244	497	7	147
rc103	10	519	7	275	516	8	262	519	8	192
rc104	10	565	9	481	574	8	324	555	7	277
rc105	10	459	8	327	478	8	181	435	6	168
rc106	10	458	11	394	481	8	254	464	8	174
rc107	10	515	9	488	514	8	258	487	7	255
rc108	10	546	11	388	536	10	296	535	8	216
rc201	10	1305	41	865	1343	37	891	1034	19	311
rc202	10	1461	34	1055	1435	37	1011	1184	18	362
rc203	10	1573	35	846	1562	32	1298	1364	19	396
rc204	10	1656	26	654	1666	26	1445	1607	19	469
rc205	10	1381	36	1136	1363	34	1012	1045	17	359
rc206	10	1495	34	820	1477	34	1288	1326	20	605
rc207	10	1531	29	873	1508	31	1518	1427	19	483
rc208	10	1606	31	1389	1610	29	1579	1583	19	579

Table 26: Results for Cordeau et al. instances for 2 tours

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
pr01	4	471	11	128	493	13	159	437	6	84
pr02	9	660	12	696	669	18	416	631	9	214
pr03	14	714	17	954	713	21	847	635	11	334
pr04	19	863	20	2197	832	20	1203	797	16	593
pr05	24	1011	32	3459	1047	23	2486	918	17	984
pr06	28	997	24	4219	964	24	2086	869	17	1149
pr07	7	552	7	422	555	10	301	552	8	137
pr08	14	796	19	1165	783	17	737	722	10	360
pr09	21	867	22	2592	816	26	1208	749	13	737
pr10	28	1004	26	4475	1058	26	2481	932	18	1380
pr11	4	542	10	155	521	10	218	528	6	135
pr12	9	727	10	641	727	16	616	686	6	367
pr13	14	757	14	1448	799	17	960	750	8	499
pr14	19	925	22	2432	943	17	1695	867	17	897
pr15	24	1126	29	6147	1101	28	3109	981	14	1453
pr16	28	1110	26	5159	1076	24	3199	929	14	1509
pr17	7	624	11	935	620	10	398	594	7	254
pr18	14	877	13	1152	892	21	1259	790	11	490
pr19	21	955	21	3242	899	18	2083	855	12	1020
pr20	28	1056	24	4568	1110	24	3003	950	15	1586

Table 27: Results for Solomon instances for 3 tours

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
c101	10	790	20	704	800	22	310	780	15	308
c102	10	890	20	664	890	19	419	880	15	541
c103	10	960	20	772	960	17	626	940	16	534
c104	10	1010	15	960	1010	16	981	1020	14	760
c105	10	840	26	615	850	20	398	830	16	361
c106	10	840	23	764	850	23	447	830	16	441
c107	10	900	19	1134	900	19	464	880	15	354
c108	10	900	22	1875	910	23	514	880	13	482
c109	10	950	16	674	950	15	694	960	15	737
c201	10	1750	35	1176	1760	27	1341	1750	22	618
c202	10	1750	38	947	1780	28	1151	1750	27	548
c203	10	1760	44	584	1750	35	1247	1750	23	583
c204	10	1780	43	525	1780	30	1579	1790	22	653
c205	10	1770	26	636	1770	24	1330	1790	21	715
c206	10	1770	21	538	1800	21	1523	1800	20	595
c207	10	1810	22	875	1790	24	1463	1780	20	641
c208	10	1810	21	883	1810	21	1721	1810	19	756
r101	10	481	17	314	475	16	167	448	11	163
r102	10	685	14	537	670	16	412	666	13	334
r103	10	720	15	878	720	14	548	708	12	365
r104	10	765	13	1138	767	15	621	746	11	488
r105	10	609	19	831	596	16	294	592	11	289
r106	10	719	13	594	704	13	403	699	13	453
r107	10	747	14	810	743	14	540	744	13	471
r108	10	790	14	1616	794	14	671	769	13	518
r109	10	699	16	1110	697	16	495	677	12	325
r110	10	711	16	694	718	16	538	707	13	491
r111	10	764	14	941	764	14	602	758	12	466
r112	10	758	14	1023	757	14	864	746	13	598
r201	10	1408	65	784	1412	57	1352	1170	29	478
r202	10	1443	59	619	1443	58	1414	1344	27	601
r203	10	1458	63	394	1458	54	1494	1387	30	705
r204	10	1458	56	311	1458	40	1711	1444	24	766
r205	10	1458	63	408	1458	60	1379	1404	29	656
r206	10	1458	52	334	1458	54	1523	1428	27	778
r207	10	1458	63	318	1458	58	1570	1453	27	654
r208	10	1458	49	309	1458	39	1673	1458	24	729
r209	10	1458	53	359	1458	52	1443	1409	28	599
r210	10	1458	56	347	1458	51	1455	1420	28	611
r211	10	1458	56	315	1458	44	1508	1458	27	607
rc101	10	604	12	592	614	9	236	614	9	216
rc102	10	698	13	1389	695	11	459	692	12	409
rc103	10	747	11	760	763	12	500	729	11	359
rc104	10	822	11	687	816	11	512	804	10	523
rc105	10	654	11	487	648	11	303	634	9	271
rc106	10	678	13	549	683	13	390	670	11	358
rc107	10	745	14	690	752	15	564	724	11	350
rc108	10	757	14	562	780	12	531	763	11	538
rc201	10	1625	53	743	1650	53	1385	1374	27	564
rc202	10	1686	58	664	1684	49	1530	1465	26	713
rc203	10	1724	42	444	1724	43	1398	1612	25	885
rc204	10	1724	42	350	1724	40	1534	1701	26	634
rc205	10	1659	54	601	1660	49	1462	1376	28	551
rc206	10	1708	43	562	1721	47	1734	1583	26	706
rc207	10	1713	39	610	1719	46	1452	1629	24	678
rc208	10	1724	51	359	1724	49	1481	1724	27	818

Table 28: Results for Cordeau et al. instances for 3 tours

		ILS			**CSCRatio**			**CSCRoutes**		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
pr01	4	598	20	84	582	12	178	582	9	180
pr02	9	899	21	799	882	19	705	821	12	407
pr03	14	946	29	1978	935	26	1355	887	17	978
pr04	19	1195	31	6425	1201	32	2121	1146	24	1607
pr05	24	1356	39	6107	1382	42	4404	1308	21	2460
pr06	28	1376	43	12652	1353	34	3979	1220	28	2398
pr07	7	713	13	898	721	16	465	693	10	265
pr08	14	1082	27	2052	1075	28	1479	960	20	674
pr09	21	1144	29	6240	1203	27	2843	1067	19	1334
pr10	28	1473	38	11768	1446	41	4984	1312	31	2708
pr11	4	632	13	76	635	11	217	617	9	132
pr12	9	902	19	756	927	16	1282	899	10	502
pr13	14	1046	20	2150	1045	22	1877	984	15	958
pr14	19	1197	29	4614	1240	32	3304	1210	22	1951
pr15	24	1488	36	6891	1477	34	5465	1426	21	3119
pr16	28	1478	36	8589	1501	29	5878	1355	23	3808
pr17	7	808	20	344	793	13	554	781	12	338
pr18	14	1165	22	2078	1133	24	1783	1050	17	964
pr19	21	1238	27	5001	1331	29	4142	1241	21	2409
pr20	28	1514	31	14826	1495	33	6340	1402	26	4079

Table 29: Results for Solomon instances for 4 tours

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
c101	10	1000	27	851	1010	23	509	960	15	576
c102	10	1090	24	1211	1110	19	802	1080	17	699
c103	10	1150	28	816	1160	25	841	1140	18	685
c104	10	1220	19	2037	1200	22	1108	1230	17	1148
c105	10	1030	26	1560	1060	23	660	1020	20	487
c106	10	1040	25	932	1060	26	733	1030	22	758
c107	10	1100	23	789	1110	20	937	1080	19	779
c108	10	1100	23	1309	1100	24	808	1080	24	875
c109	10	1180	22	1323	1160	20	1207	1160	19	997
c201	10	1810	40	318	1810	45	1408	1810	32	788
c202	10	1810	57	302	1810	43	1436	1810	32	740
c203	10	1810	51	303	1810	40	1455	1810	29	802
c204	10	1810	56	293	1810	40	1565	1810	30	823
c205	10	1810	51	297	1810	40	1446	1810	33	804
c206	10	1810	55	257	1810	37	1450	1810	32	788
c207	10	1810	53	290	1810	39	1439	1810	33	810
c208	10	1810	48	286	1810	40	1453	1810	32	768
r101	10	601	19	869	598	18	248	576	13	268
r102	10	807	22	1008	806	20	659	794	15	561
r103	10	878	22	862	888	22	894	858	14	603
r104	10	941	23	1282	951	16	921	925	17	800
r105	10	735	24	679	753	22	674	712	16	420
r106	10	870	20	795	881	20	907	845	17	675
r107	10	927	21	1018	912	19	1054	907	16	785
r108	10	982	17	1071	967	18	980	959	17	809
r109	10	866	18	1148	874	22	920	854	16	843
r110	10	870	22	1048	879	20	815	867	15	736
r111	10	935	20	2453	924	20	1059	905	16	717
r112	10	939	18	2248	949	20	1301	933	17	1033
r201	10	1458	75	403	1458	70	1527	1339	36	966
r202	10	1458	65	340	1458	58	1646	1392	37	775
r203	10	1458	60	284	1458	56	1636	1456	32	879
r204	10	1458	57	198	1458	40	1714	1458	30	831
r205	10	1458	69	259	1458	63	1591	1451	32	848
r206	10	1458	62	219	1458	60	1547	1458	36	1106
r207	10	1458	67	186	1458	54	1591	1458	34	802
r208	10	1458	49	133	1458	31	1551	1458	28	876
r209	10	1458	59	214	1458	49	1691	1458	34	870
r210	10	1458	66	270	1458	62	1626	1458	32	751
r211	10	1458	56	186	1458	51	1395	1458	34	791
rc101	10	794	16	1098	779	13	348	776	11	439
rc102	10	881	19	1380	878	13	615	855	14	421
rc103	10	947	15	1060	965	13	841	931	13	553
rc104	10	1019	14	1605	1024	16	797	1014	13	652
rc105	10	841	16	734	826	15	494	814	14	416
rc106	10	874	16	879	866	14	703	852	13	538
rc107	10	951	15	1147	949	15	590	956	12	624
rc108	10	998	14	2110	988	18	811	975	16	877
rc201	10	1724	63	427	1724	65	1562	1587	34	839
rc202	10	1724	67	342	1724	57	1489	1640	34	1019
rc203	10	1724	63	287	1724	49	1601	1719	33	1202
rc204	10	1724	49	201	1724	40	1524	1724	33	829
rc205	10	1724	59	364	1724	56	1561	1593	31	720
rc206	10	1724	59	295	1724	65	1526	1724	31	838
rc207	10	1724	57	300	1724	61	1553	1718	30	1026
rc208	10	1724	55	297	1724	53	1627	1724	31	805

Table 30: Results for Cordeau et al. instances for 4 tours

		ILS			CSCRatio			CSCRoutes		
Name	Clusters	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
pr01	4	644	17	72	654	17	297	649	10	203
pr02	9	1014	23	807	1020	20	1303	947	16	826
pr03	14	1162	27	3153	1181	26	2727	1087	21	1346
pr04	19	1452	40	6826	1453	37	3245	1343	27	2713
pr05	24	1665	56	19036	1670	49	5821	1588	39	3559
pr06	28	1696	50	10613	1711	48	6374	1544	32	5576
pr07	7	840	19	436	834	20	576	800	12	511
pr08	14	1267	37	2392	1272	34	2393	1199	23	1064
pr09	21	1460	43	6910	1507	46	4527	1421	30	2812
pr10	28	1782	56	11975	1785	50	9202	1614	39	5001
pr11	4	654	17	57	654	17	234	654	13	209
pr12	9	1041	23	761	1054	21	1723	1056	16	812
pr13	14	1263	33	3564	1255	34	2307	1157	21	1269
pr14	19	1528	34	8541	1583	37	5981	1461	25	2948
pr15	24	1818	52	9163	1805	41	7763	1658	24	4599
pr16	28	1889	40	16763	1870	41	12066	1740	26	6058
pr17	7	889	22	291	881	18	619	873	13	404
pr18	14	1352	28	3032	1384	29	2590	1318	23	1663
pr19	21	1560	39	7442	1636	34	6206	1487	27	3980
pr20	28	1846	47	13714	1867	46	8963	1784	35	6845

Table 31: Results for t1* instances

			ILS			CSCRatio			CSCRoutes		
Name	Clusters	Tours	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
t101	10	1	387	9	267	387	8	241	375	7	168
t102	10	2	772	10	593	763	11	905	766	11	455
t103	16	2	786	10	1177	803	13	1143	797	12	806
t104	12	2	737	9	824	748	10	744	739	9	497
t105	15	1	433	8	329	428	7	384	433	8	378
t106	18	3	1167	16	2205	1179	16	3039	1164	16	1788
t107	13	2	787	15	1377	759	12	883	773	14	769
t108	15	2	711	16	1428	708	15	1028	712	16	868
t109	10	3	1114	16	1516	1097	14	1305	1069	16	815
t110	16	2	807	14	1494	817	15	1132	809	15	1260
t111	19	2	821	15	1845	812	14	1257	815	15	1150
t112	14	2	800	14	740	793	13	961	821	14	867
t113	15	3	1091	19	2766	1065	17	1672	1058	17	1257
t114	12	1	467	7	334	476	7	409	456	8	277
t115	10	3	1059	12	1207	1062	14	1425	1035	13	688
t116	18	2	840	14	889	841	15	1260	839	11	1020
t117	19	1	452	9	725	446	6	638	462	8	511
t118	15	3	1140	23	1615	1133	16	1925	1140	21	1726
t119	18	3	1163	25	1732	1142	21	2416	1159	20	2214
t120	17	2	1023	15	2902	1014	16	1951	1002	14	1038
t121	16	1	424	8	302	445	6	481	428	4	331
t122	14	1	468	4	751	469	4	449	470	2	242
t123	15	1	404	5	247	410	7	339	409	8	293
t124	12	1	435	5	180	467	7	337	471	4	222
t125	18	3	1176	19	2298	1169	12	2180	1179	14	1556
t126	12	1	413	5	294	414	6	319	408	4	208
t127	11	3	1025	15	1595	1023	13	1152	1012	14	846
t128	14	3	1111	22	1994	1105	22	2027	1062	23	1273
t129	13	1	432	7	263	442	6	329	441	7	263
t130	18	2	812	18	956	804	15	1178	764	12	1023
t131	16	1	400	7	322	407	9	397	365	7	263
t132	19	1	420	10	497	416	7	509	418	7	458
t133	13	2	798	12	1395	804	17	854	804	14	660
t134	18	3	1212	24	3719	1227	24	2220	1233	21	2323
t135	16	2	823	15	2377	780	15	1083	801	15	922
t136	10	2	756	8	456	762	11	568	746	10	414
t137	17	3	1119	18	2254	1089	20	1758	1069	16	1430
t138	17	3	1222	19	2378	1203	18	2080	1206	18	2108
t139	15	3	1115	20	2257	1154	20	2149	1138	17	1500
t140	10	3	993	18	770	1035	13	1217	1004	13	660
t141	10	2	724	13	664	746	15	682	730	14	426
t142	18	3	1185	24	2516	1178	18	3289	1166	22	1907
t143	15	1	413	8	254	416	8	444	416	9	379
t144	17	2	763	14	1222	733	15	971	729	13	701
t145	10	1	357	9	196	370	6	289	364	6	151
t146	13	2	767	13	960	773	11	862	768	14	586
t147	13	3	1078	15	2124	1103	14	1306	1085	20	1288
t148	14	1	468	5	256	475	5	423	475	5	250
t149	13	3	1072	21	2070	1065	16	1230	1084	15	1214
t150	16	1	487	6	477	485	6	528	487	6	399

Table 32: Results for t2* instances

			ILS			**CSCRatio**			**CSCRoutes**		
Name	Clusters	Tours	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)	Profit	Transf	CPU(ms)
t201	12	1	183	3	76	187	3	107	177	2	102
t202	14	1	193	2	102	193	2	111	193	2	111
t203	10	1	174	3	67	179	4	120	178	2	95
t204	14	1	171	3	153	171	5	126	171	3	137
t205	13	3	447	10	1070	434	10	267	445	10	345
t206	17	1	196	3	212	197	7	165	196	6	201
t207	14	1	174	2	89	174	2	101	201	3	111
t208	19	1	162	3	90	176	3	138	176	3	157
t209	17	3	455	13	1010	464	13	552	446	9	480
t210	20	3	481	11	1086	482	9	527	497	9	666
t211	12	3	472	7	703	473	8	374	474	7	462
t212	10	3	461	4	394	466	5	260	460	5	234
t213	17	3	498	11	825	500	14	644	490	9	681
t214	14	2	310	6	380	322	8	283	317	6	249
t215	13	3	424	10	429	425	11	358	424	12	381
t216	12	3	463	11	600	462	10	357	468	11	328
t217	11	3	463	8	821	463	7	242	462	7	342
t218	10	1	155	1	50	155	1	69	155	2	70
t219	16	3	473	11	761	482	15	638	483	12	692
t220	13	2	334	5	592	347	4	237	329	5	185
t221	18	2	280	10	315	287	9	285	280	6	323
t222	19	2	396	10	520	396	12	456	389	11	441
t223	15	1	183	4	65	216	8	158	229	6	162
t224	13	3	402	5	539	406	6	301	405	5	282
t225	18	3	543	14	1623	551	13	648	549	12	845
t226	19	3	569	14	1151	563	14	741	578	13	819
t227	13	1	159	3	73	159	3	100	159	3	115
t228	16	3	537	11	812	531	11	697	530	11	626
t229	19	1	178	4	123	178	4	157	174	3	176
t230	11	2	288	7	176	286	11	147	285	8	193
t231	11	3	498	8	426	501	8	507	489	8	343
t232	17	3	522	15	1309	535	11	642	525	13	655
t233	15	1	180	5	112	209	6	145	213	7	163
t234	15	3	488	8	888	501	8	526	503	10	434
t235	14	3	484	12	468	496	13	366	482	11	430
t236	15	1	175	3	84	174	3	121	175	3	132
t237	11	3	473	10	954	477	9	489	479	10	491
t238	12	3	526	8	731	522	8	485	533	7	439
t239	19	3	508	15	1703	509	12	771	508	13	914
t240	12	2	297	6	183	308	8	190	322	8	232
t241	14	1	170	3	48	172	4	98	172	3	113
t242	10	1	180	2	89	180	2	91	180	2	83
t243	15	1	170	2	77	195	5	136	201	4	149
t244	10	2	331	9	176	331	8	218	332	7	183
t245	14	2	291	5	168	299	5	136	285	7	182
t246	15	3	455	11	548	453	8	459	452	9	436
t247	13	3	445	10	582	454	9	370	444	9	422
t248	17	3	445	13	941	467	13	817	465	14	629
t249	11	3	431	10	474	438	12	226	425	6	231
t250	20	1	200	7	276	201	7	174	192	5	202