*STAIRS 2014*
*U. Endriss and J. Leite (Eds.)*

*doi:10.3233/978-1-61499-421-3-260*

# Electronic Tourist Guides: User-friendly Editing of Automatically Planned Routes

Richard SCHALLER [a]
[a] *E-mail: richard.schaller@fau.de*
*AI Group, University of Erlangen-Nuremberg, Germany*

**Abstract.** Sightseeing tourists on a city trip can be supported by an electronic tourist guide. More advanced systems generate personalized routes taking into account user preferences and traveling distances between sights. Recommenders can be used in order to estimate user's interest in the available sights. Typically recommendations are then fed into a planning algorithm that tries to determine a subset of sights and a suitable order. The aim being to include the most interesting sites for the user and possibly as many interesting sights as possible within a given set of constraints. Additional constraints can be considered such as a sight being a must-see for the user or opening hours of sights. At some point the generated route is shown to the user. If the user is not satisfied with the route he may want to change it. In most systems modifying the constraints or user preferences and then regenerating a completely new route is the only option to accomplish this task. More fine-grained control over route modifications is in typically unavailable.
In this paper we examine how sightseeing tourists can be supported in editing an automatically generated route. We discuss what is necessary for editing operations to meet user's expectations. We then show how six different route editing operations can be implemented via the idea of collapsing and expanding a route. Finally we present a potential route editing user interface for an electronic tourist guide. The presented interface makes disadvantageous route modifications obvious by providing instant feedback on how an operation affects the route.

## 1. Introduction

When planning a city trip tourists are faced with various tasks: a city and time frame has to be chosen, various traveling options and hotels have to be considered and finally the stay itself has to be planned. This last aspect is particularly challenging for trips to bigger cities with hundreds of sights, restaurants, shopping locations and various other tourist-related venues. These Points of Interest (POIs) tend to be geographically (e.g. throughout a city) and temporally (e.g. different opening hours during the day) dispersed. Tourists usually consider many different properties and contextual factors when deciding for which POIs to visit and in which order, i.e. the type, price, location and opening hours of the POI, the distance/transfer links to the POI and the personal interest in the POI but also general context factors such as time of day or weather [1]. An electronic tourist guide can assist visitors in finding optimal or near-optimal solutions to this multi-criteria optimization problem. In [2] three components of an electronic tourist guide are described: a recommendation phase that facilitates a user profile and POI data to generate a list of

personalized POI recommendations. In a second step, a planning algorithm determines an optimal subset and order of these POIs taking into account various constraints such as opening hours, traveling times or user restrictions such as start and end time of the route. As there might be user preferences or contextual factors not being modeled in the system it is necessary to have a third step which allows modifications of the generated route, like removal or insertion of a POI visit. Also during plan execution it is more than likely that visitors will need to re-plan due to external or internal causes, e.g. a POI being overcrowded or the tourist being exhausted. In [3] an electronic tourist guide was evaluated in a field trial: only 55% of the actually visited sights were contained in the route planed beforehand showing a need for route modifications on the move.

Most research performed on electronic tourist guides focuses on either the recommendation of POIs or on the planning of routes. The resulting routes are packed – as this is one of the objectives of the planning algorithm – making route modifications in most cases impossible without rendering the complete route infeasible. Consider, for example, adding a POI: if a gap in the route were big enough to insert a POI, the planning algorithm would already have done that. Most likely a visitor would expect to reduce the visit duration of other POIs to make room for the additional POI. On the other hand visitors would like to close gaps after removing POIs by expanding the visit of the remaining POIs. Additionally, moving POIs within a route might need to shorten visits as traveling times likely increase due to deriving from the optimal ordering.

In this paper we will focus on operations that can be performed on an already given route making it easy for visitors to modify routes to match their preferences:

We introduce the concept of collapsing the visit durations in a route to make room for further modifications. We furthermore, introduce the corresponding concept of expanding a route by increasing visit durations to fill gaps. We then show how user modifications can be implemented by making use of collapsing and expanding a route. Next we discuss removing, inserting and moving a POI, changing the visit duration of a POI and filling up a route with additional POIs. Finally we present a system where we implemented these ideas.

## 2. Related Work

**Recommenders.** Although this paper focuses on supporting route modification it is worthwhile to explore how recommendations can be obtained as they can be used for filling up a route with additional POIs. There are two main types of recommenders used in electronic tourist guides. These are content-based recommenders and collaborative filtering approaches. For content-based recommenders additional data for each POI and a user profile for each user have to be acquired. For example in [4] an ontology is used for categorizing each POI, e.g. as memorial. The user is then asked upon the first use of the system to specify his interest in these categories resulting in a detailed user profile. A semantic matching is then used to find those POIs that are most similar to the user's profile. Contrastingly, collaborative filtering approaches use other users' POI selections to learn which POIs are similar [5]: users with similar tastes are also likely to select the same POIs. Hybrid recommenders combine both approaches to improve the quality of recommendations [5,6]. An overview and comparison of the use of recommender systems in the field of electronic tourist guides is given in [1].

**Planning.** For planning purposes the optimization problem has to be formalized. [7] suggests to use the OP (Orienteering Problem) as the formal framework, a derivative of the Traveling Salesman Problem (TSP): a set of locations with an assigned score and information about traveling times between all locations is given. Additionally a starting point and time as well as a destination point and time is given. A solution of the OP is a route that contains a subset of the available locations and connects the starting point with the ending point. The optimal solution is the route that maximizes the sum of the collected scores. The OP can be extend to consider various additional constraints such as opening hours (Orienteering Problem with Time Windows) or to plan for multiple days (Team Orienteering Problem). For solving the OP both exact [8] and heuristic [4] approaches exist. In [7] an Iterative Local Search is used to solve the OP: a route is created by iteratively adding and removing POIs. Choosing which POI to add considers the score of the POI and the additional required time caused by the POI insertion. Adding POIs is done until no further POIs can be added and the algorithm is stuck in a (local) optima. Then removing of POIs is performed to escape the local optima. Under all circumstances the route is feasible, meaning start and end time constraints, traveling times and visit durations are considered and additional constraints such as opening hours are respected.

**Electronic Tourist Guides.** Various research projects have developed prototypical or complete electronic tourist guides: The CT-Planner described in [9] uses a content-based recommender to calculate scores for all POIs. For route generation a heuristic approximation algorithm is used. To obtain a user profile a user can specify his interests directly or indirectly by choosing between different routes. The latter are generated by using the recommender with a slightly modified user profile. In case the user decides for one of these alternatives the user profile is updated accordingly. Furthermore the user can influence the route generation by selecting certain POIs as must-haves or don'ts. However, there is no possibility to edit the generated route in detail. The system developed in [2] uses a content-based recommender to choose POIs but also let users specify which POIs are of interest to them. The planning is based on an Iterative Local Search algorithm. To customize the generated route various operations such as adding, removing or moving a POI are provided. These operations shift beginning times of all POIs after the editing position accordingly. It is possible for the user to make modifications that violate some of the restrictions. From the description given in [2] it does not become clear whether this applies only to user specified restrictions such as the end time or to all kinds of restrictions. Violating POI restrictions such as opening hours would render routes infeasible. Thus it is plausible that such modifications are forbidden. Permitted modification will shift visits beyond the specified end time as needed, effectively removing this constraint. To our knowledge, visit durations are not adjusted.

## 3. User-friendly editing operations

Considering the systems presented in the previous section most do either provide no route modifications or do so via recalculating a route under modified constraints. A fine-grained control of routes is only provided by [2] where the route end time constraint is lifted. Depending on the type of POIs this lifting might not be enough to yield still feasible routes. Consider for example POIs with very limited opening hours where shifting results in missing the opening hours. The user is either left behind with no explanation

why his operation is not permitted or a complex explanation – probably involving multiple constraints – must be generated and communicated.
We instead suppose to lift a different constraint: the visit duration of POIs. In most systems its value is either fixed per POI or estimated in advance for each user. We argue that the visit duration is a soft constraint from a user's point of view and the user is flexible regarding the exact visit duration: routes do not suddenly become infeasible if a museum visit is scheduled for a few minutes less. Of course further editing might reduce the visit duration even more and make the route unrealistic. However, our approach has the advantage that the user can see how the route slowing becomes more and more infeasible and receives feedback regarding why his editing operations are unfavorable. Thus we suggest to adjust the visit durations of other POIs in order to allow editing operations under most circumstances. The only exception where editing operations are still forbidden is when visit durations cannot be shortened further or route independent conflicts make the operation fail, e.g. visiting two POIs at the same time at different locations.
As the visit duration becomes very flexible and time can be shuffled arbitrary between POIs we make some assumptions regarding how the system should behave when performing route modifications:

- The route should always be feasible meaning all constraints are met (except the visit duration constraint): traveling times are considered, opening hours are respected, etc.
- The order of the POI remains unchanged except for the POI that is involved in the modification.
- If there are gaps in the route they should be reduced as much as possible.
- With each POI a minimum visit duration is associated which should be respected unless this results in a route modification becoming inexecutable. In this case the visit duration might be reduced further still.
- The time surplus caused by gaps should be distributed "fairly" between available POIs, meaning that time is distributed equally among all POIs. If a POI's visit duration cannot be extended any further the remaining time is distributed among the other POIs.
- Also if time is needed to perform an operation, visit duration of all POIs should be reduced in a "fair" manner. As minimum visit durations should be respected, time should be taken first from those POIs which are scheduled for a longer stay. We consider the minimum visit duration as sufficient and any additional time spent (=extra visit duration[1]) as not contributing to visitor's experience. Nevertheless we want to keep the extra visit time of all POIs equal which leads to first shortening that POI which has the most extra visit duration.

With these assumptions we can now redefine the route operations from [2]: *Remove a POI*, *Insert a POI at a specific position*, *Insert a POI at its best position* and *Move a POI*. Additionally we allow the user to *Change the visit duration of a POI*. In case the user removes multiple POIs he might like to *Fill-Up the route with POIs* without specifying the POIs to add but instead let the system decide which POIs to use.

[1] visit duration = minimum visit duration + extra visit duration

## 4. Collapsing and Expanding a Route

The different editing operations that we are going to implement either consume time or leave additional time after they were performed. For the former we want to provide as much flexibility as possible which means that we want to reduce the time spent on POIs as much as possible. We call this process of transforming a given route into a route, where every POI is only visited for no longer than its respective minimum visit duration, *collapsing a route*. The delta between the old visit duration and the new visit duration is remembered. The following pseudo code shows this function:

```
Collapse(route):
  FORALL p in route:
    diff = max(0, p.duration - p.minDuration)
    p.delta += diff
    newDuration = p.duration - diff
    Change duration of p to newDuration
```

Of course changing the visit duration of a POI results in shifting all subsequent POI visits accordingly. Shifting POI visits is a crucial part of the Iterative Local Search approach for solving the OP. Making it efficient and also taking into account restrictions such as opening hours is described in [7].

As mentioned before we make the assumption that the minimum visit duration is respected but there might be situations where we allow shorter visit durations. We therefore define a *total collapsing of a route*, which reduces the visit duration of each POI to a small fixed value – 1 min in our case[2] – as follows:

```
TotallyCollapse(route):
  FORALL p in route:
    diff = max(0, p.duration - 1)
    p.delta += diff
    newDuration = p.duration - diff
    Change duration of p to newDuration
```

The inverse operation of collapsing a route is *expanding a route*. To restore a previously collapsed route we first use the saved delta and distribute the available time between those POI visits that were previously shortened (delta $> 0$). If there are still gaps in the route the remaining time is distributed equal among all POIs. Thus expanding is split up into two steps, which differ only in the POIs being affected:

```
Expand(route):
  ExpandHelper(route,useOnlyDelta=True)
  ExpandHelper(route,useOnlyDelta=False)
```

Even though it is possible to calculate for each POI by how much its visit duration has to be extended and update them at a stroke we instead opted for incrementally expanding a route. This simplifies the process and makes it easy to take account of additional constraints, e.g. opening hours. Expanding is performed by successively increasing the visit duration of each POI one after another by a small amount – we use a step size of 1 min. For every expansion of a POI visit its corresponding delta is reduced. In case a POI visit becomes stuck and cannot be expanded further – again opening hours being a potential cause – we skip this POI and continue with the next one. After all POIs have been updated we start all over again until no further expansions are possible.

The following pseudo code shows how expanding a route is performed. Note the use of `useOnlyDelta` to select whether all POIs should be affected or only those with a saved delta:

[2] We choose 1 min as we do not want to confuse visitors with 0 min visits

```
ExpandHelper(route, useOnlyDelta):
  DO:
    expanded = False
    FORALL p in route:
      IF useOnlyDelta AND p.delta == 0:
        CONTINUE
      IF duration increase by 1 feasible for p:
        newDuration = p.duration + 1
        Change duration of p to newDuration
        p.delta = max(0, p.delta-1)
        expanded = True
  WHILE expanded
```

One caveat of the presented method is that POIs towards the start of a route are considered first. In case only a small amount of time becomes available it is given to those POIs. If this happens repeatedly instead of all time becoming available at once[3], time is not fairly split between those POIs towards the beginning and those towards the end of the route. To remedy this problem one may remember at which POI the last expansion stopped and continue from there instead of the first POI.

Consider what happens if a route is collapsed, then some operations are performed on it and then it is expanded again. There are three cases of how available time changes depending on the performed operations: a) Same amount of time is used: with the first run of `ExpandHelper` the original visit durations are restored. b) Some time was freed up: again the original visit durations are restored as above. But there is still some time left, which is distributed equally by the second run of `ExpandHelper`. c) Some time was used up: with the first run of `ExpandHelper` the original visit durations could not be restored completely, but the available time is again distributed equally. This means that POI visits that were reduced by a larger amount during collapsing than others are those whose time is partially taken away to account for the used up time. They are not fully expanded any further. The last case also means that delta remains $> 0$ for some of the POI visits. In order to not affect further editing operations it is necessary to reset it.

Of course expanding a route can also be used to reduce gaps in a newly generated route: planning algorithms may schedule POIs with their respective minimum visit duration and the resulting route is then expanded afterwards.

## 5. Route Editing Operations

All editing operations described in this section follow a general scheme: (totally) collapsing the route, performing the actual edit and finally expanding the route. Sometimes slight variations of this scheme are used to realize the provided editing operations.

**Remove POI.** As removing POIs can only result in time being freed up, collapsing the route can be skipped and expanding the route will close the resulting gap.

**Insert POI at a specific position.** For insertion of an POI at a specific position we first totally collapse the route, then insert the POI and finally expand the route:

```
Insert(route, poi, position):
  ResetDelta(route)
  TotallyCollapse(route)
  Insert poi at position
  Expand(route)
```

Totally collapsing the route might result in POI visits dropping below their minimum

[3] i.e. when a user can modify the visit duration interactively via a slider interface (see Section 6)

visit duration but this is either fixed when expanding the route (because other POIs' extra visit duration can be used) or it is necessary and there is no way around it as time has to be freed up for the added POI.

**Insert POI at its best position.** If the system should determine the best insertion position for a POI, we initially want to consider only positions which would not result in other POI visits dropping below their minimum visit duration. Thus we start with collapsing the route to their POIs minimum visit duration. Then we try to insert the POI. If this is not possible we must shorten visits below their minimum visit duration with a total collapse of the route and try the insertion of the POI again. In any case the route is expanded afterwards:

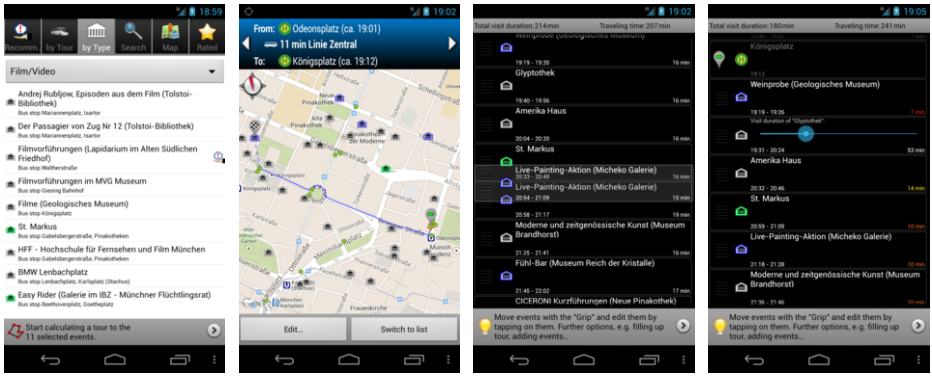
```
Insert(route, poi)
  ResetDelta(route)
  Collapse(route)
  p = find best insertion position for poi
  IF p is valid:
    Insert poi at p
  ELSE:
    TotallyCollapse(route)
    p = find best insertion position for poi
    IF p is valid:
      Insert poi at p
  Expand(route)
```

**Move POI.** When moving a POI we adhere to the general scheme of total collapsing, performing the move and expanding.

**POI visit duration change.** Changing the duration of a visit depends on whether it is shrunk or extended. In case of a shrunk collapsing the route is not necessary because no additional time is needed. As the duration of the affected POI is set directly it must be excluded from the following expanding of the route (lock). Otherwise its visit duration might get re-increased. In case a visit duration is extended we first totally collapse the route. Then we change the visit duration. If the requested new duration of the POI is not feasible we only change it to the maximum feasible value. Again we have to make sure that the expanding of the route will leave out the modified POI:

```
ChangeDuration(route, position, newDuration):
  poi = route[position]
  oldDuration = poi.duration
  IF newDuration < oldDuration:
    ResetDelta(route)
    Change duration of poi to newDuration
    lock poi.duration
    Expand(route)
    unlock poi.duration
  ELSE IF newDuration > oldDuration:
    ResetDelta(route)
    TotallyCollapse(route)
    maxDuration = max feasible duration of poi
    d = min(maxDuration, newDuration)
    Change duration of poi to d
    lock poi.duration
    Expand(route)
    unlock poi.duration
```

**Fill-up route.** Routes with larger gaps might be filled up by letting the system decide which POIs to add. The systems choice might be restricted to a set of POIs – a candidate set. Recommenders could be used to determine the set and/or to weight POIs. For filling up a route we first collapse all visits to their minimum visit duration. Then we perform one step of the Iterative Local Search algorithm for the OP described in [7]: we iteratively




(a) (b) (c) (d)

**Figure 1.** (a) Browsing through the available events by different means. (b) Following a generated route on the map. (c) Editing the route by moving an event. (d) Editing the route by changing the visit duration.

insert one POI after another until the route is too packed to add further POIs. In each iteration we determine the best candidate and its best position. We then add this POI at that position to the route and remove it from the candidate set. Afterwards we expand the route again to close any remaining gaps:

```
FillUp(route, poiSet)
  ResetDelta(route)
  Collapse(route)
  DO:
    insert best POI of poiSet at best position
    IF route was changed
      remove inserted POI from poiSet
  WHILE route was changed
  Expand(route)
```

Our local search approach on filling up a route results in a local optimum to be returned which might be unfortunate. However, in our experience this seems to be less problematic as long as the route to be filled-up is not empty or near empty: the existing POIs reduce the search space, set a general frame for the resulting route and thus make sure that the generated plan is plausible.

## 6. Route Editing User Interface

The editing operations described in the previous section can be implemented via a user interface designed to assist visitors in modifying a generated route. We implemented an electronic tourist guide for a special kind of tourism: visiting a distributed event. A distributed event [10] is a collection of smaller, single events occurring at approximately the same time and conforming to one overarching theme. Well known examples include the Cannes International Film Festival, the Edinburgh Festival Fringe, and Montreal International Jazz Festival. We focus in this paper on the Long Night of Munich Museums. What many of these events have in common is that they have huge number of diverse sub-events that are geographically and temporally dispersed. Thus visitors of distributed events are facing very similar problems to those visiting a city for sightseeing purposes. One main difference are the special shuttle buses that are provided and that connect the different events. In the developed Android app we provide assistance for three components of electronic tourist guides mentioned in [2]:

In a first step we determine a set of POIs of potential interest to the visitor. For this purpose we use a recommender system [5] but do not feed the results into the route planning algorithm. Instead the recommendations are shown to the user to allow him to decide upon the events he is interested in. This also makes it possible to provide users other means of accessing events [11]. Each tab in Figure 1a provides one of these options: the user can browse events organized geographically by their position on bus routes, by type, by searching for their name/description or by selecting them on a map. The "Rated" tab shows a list of already selected events.
In a next step we generate routes containing as many events as possible from the set of selected events. This is done with an Iterative Local Search algorithm similar to [7]. The resulting route presented to the user as a list or on a map (see Figure 1b).
Finally when the user decides to modify the route we support him on a dedicated editing screen (see Figure 1c and Figure 1d): all the editing operations described in Section 5 are available: when adding events the user is directed to the already known event browser (see Figure 1a) to choose an event. Moving an event is performed by drag and drop as depicted in Figure 1c. During the dragging of an event the route is continuously updated to reflect the current position of the event in the route. For changing the event visit duration we use a slider interface as shown in Figure 1d: dragging the slider to the right instantaneously increases the visit duration of the selected event and decreases those of the other events. As for each slider movement the complete route has to be collapsed and expanded we experienced some performance problems on older low-end smartphones. To solve these problems and provide a responsive user interface we increased the step size to 5 min when expanding the route. This only partially expands the route and would leave gaps of up to 4 min which we close by expanding again with a 1 min step size.
At the top of the editing screen the total time spent at events and the total traveling time is shown. This helps users in judging the effects of an editing operation on the route as a whole. Additionally event visits dropping below the minimum visit duration are visually highlighted via color depending on how much they fall short of the desired visit duration. This makes it easy for the user to realize that this constraint is violated and might encourage him to make further route modifications to remedy this problem.

## 7. Conclusion and Future Work

In this paper we have provided insights into how sightseeing tourists could be supported in editing an automatically generated route. We first looked into why it is desirable to give users the opportunity to modify a route instead of focusing on improvements to the POI recommender or the route planning algorithms. We also gave an overview on how current systems approach this need showing that all but one system rely on re-planning. Only that system provides fine-grained control on route modifications to the user but ignores the user-specified route end time. We argued that this is neither sufficient to allow certain route modifications in case POIs have opening hours nor do users expect the system to dismiss their input. Thus we next made up guidelines into how an editing system should behave to meet users' expectations such as a fair distribution of time surplus. We finally set on six operations we want the system to support. To define these we established and defined the concept of collapsing and expanding a route. Collapsing is reducing all POI visits to the bare minimum while expanding a route is the inverse

operation and furthermore closes all remaining gaps in the route. With the help of these we then provided solutions for the six operations which adhere to the guidelines made up earlier. We showed that most operations consist of the three steps collapsing the route, performing edit operation and then expanding the route again. Finally we looked into an assistance system for distributed events and especially into the route editing user interface that uses the operations developed in this paper. The interface provides instant feedback on how an operation affects the route making disadvantageous route modifications obvious while not hindering users in performing the modifications they envisioned.
Even though some of the operations make use of ideas used by planning algorithms, the presented solutions are independent of these and can be used in conjunction with any route generation algorithm. Nevertheless, a tighter integration between planning and route modifications is imaginable: the fill-up algorithm presented in this paper is a first step in this direction. More advanced route improvement strategies could recommend advantageous editing operation sequences such as: "removing POI A, switching POI B and C and then adding POI D, E and F would save you a lot of traveling and would increase the total time spend at POIs." This would show the user a way back from a customized route to an optimized route.
In the near future we plan to evaluate how the editing interface is used in a field trial. We are interested in how often the available operations are used, what events they are used on and in which way the route is modified. We plan to use metrics, such as the topical diversity of route events, to measure how routes before and after the edit differ from each other. This might gain insights into what properties a route generation algorithm should take into account. If user-made route improvements can be measured it might also be possible to use these metrics during the planning phase to judge different route options.

## References

[1] Katerina Kabassi. Personalizing recommendations for tourists. *Telematics and Informatics*, 27(1):51–66, 2010.

[2] Ander Garcia, Olatz Arbelaitz, Maria Teresa Linaza, Pieter Vansteenwegen, and Wouter Souffriau. Personalized tourist route generation. In *Current Trends in Web Engineering*, pages 486–497. Springer, 2010.

[3] Ronny Kramer, Marko Modsching, Klaus Hagen, and Ulrike Gretzel. Behavioural impacts of mobile tour guides. *ENTER*, pages 109–118, 2007.

[4] Ronny Kramer, Marko Modsching, and Klaus Ten Hagen. A city guide agent creating and adapting individual sightseeing tours based on field trial results. *International Journal of Computational Intelligence Research*, 2(2):191–206, 2006.

[5] Richard Schaller, Morgan Harvey, and David Elsweiler. RecSys for distributed events: Investigating the influence of recommendations on visitor plans. In *Proc. of SIGIR*, 2013.

[6] Eui-young Kang, Hanil Kim, and Jungwon Cho. Personalization method for tourist point of interest (POI) recommendation. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 392–400. Springer, 2006.

[7] Pieter Vansteenwegen. Planning in tourism and public transportation. *4OR*, 7(3):293–296, 2009.

[8] Marcus Poggi, Henrique Viana, and Eduardo Uchoa. The team orienteering problem: Formulations and branch-cut and price. In *10th Workshop on ATMOS*, page 142, 2010.

[9] Yohei Kurata. CT-Planner2: More flexible and interactive assistance for day tour planning. In *Information and Communication Technologies in Tourism 2011*, pages 25–37. Springer, 2011.

[10] Richard Schaller. Planning and navigational assistance for distributed events. In *Proceedings of the 2nd Workshop on Context Aware Intelligent Assistance*, 2011.

[11] Richard Schaller, Morgan Harvey, and David Elsweiler. Entertainment on the go: finding things to do and see while visiting distributed events. In *Proc. IIiX*, pages 90–99, 2012.