Planning for Tourism Routes using Social Networks

Isabel Cenamor, Sergio Nuñez, Tomás de la Rosa and Daniel Borrajo

Computer Science Department, Universidad Carlos III de Madrid. Leganés (Madrid) Spain

Abstract

Traveling recommendation systems have become very popular applications for organizing and planning tourist trips. Among other challenges, these applications are faced with the task of maintaining updated information about popular tourist destinations, as well as providing useful tourist guides that meet the users preferences. In this work we present the PlanTour, a system that creates personalized tourist plans using the human-generated information gathered from the minube[1] traveling social network. The system follows an automated planning approach to generate a multiple-day plan with the most relevant points of interest of the city/region being visited. Particularly, the system collects information of users and points of interest from minube, groups these points with clustering techniques to split the problem into per-day sub-problems. Then, it uses an off-the-shelf domain-independent automated planner that finds good quality tourist plans. Unlike other tourist recommender systems, the PlanTour planner is able to organize relevant points of interest taking into account user's expected drives, and user scores from a real social network. The paper also highlights how to use human provided recommendations to guide the search for solutions of combinatorial tasks. The resulting intelligent system opens new possibilities of combining human-generated knowledge with efficient automated techniques when solving hard computational tasks. From an engineering perspective we advocate for the use of declarative representations of problem solving tasks that have been shown to improve modeling and maintenance of intelligent

[1] www.minube.com

systems.

Keywords: Tourism Routes, Automated Planning, Recommender Systems

1. Introduction

Tourism is an important social, cultural and economic phenomenon that includes the movement of millions of people around the world with a big impact on the economy of many countries. Therefore, the generation of tourism-related tools can have a huge impact in society. Traveling recommendation systems have become very popular applications for organizing and planning tourist trips [1, 2, 3, 4]. One of the main bottlenecks of this type of systems consists of the initial population and later maintenance of the information about Points Of Interest (POIs), user ratings, and connection with geographic systems. However, in recent years we have seen the emergence of new social network platforms where users can easily and are willing to update that kind of information (e.g. TripAdvisor[2] or MINUBE[3]). Also, the extensive use of tourist mobile applications allows users to request real time information about the schedules, guides or plans that fulfill their preferences [5]. Data might come from different services, so middle layers should be developed, as wrappers and crawlers that obtain and integrate available data.

From a crowdsourcing perspective [6], users of traveling social networks do not receive an explicit call for supplying relevant tourist information or composing a plan. Instead, they are encouraged to share their experience of past trips and give recommendations to everyone. Therefore, users are helping to acquire personalized relevant information using a collaborative filtering mechanism [7]. Collaborative filtering provides a subset of recommendations on what to visit, where to sleep or where to eat. Additionally, the network structure facilitates the acquisition of personalized information related to user's contacts, which can greatly help on weighting the recommendations by closeness to the user.

[2] `www.tripadvisor.com`
[3] `www.minube.com`

As in other application areas, once the data collection and maintenance of information has a reasonable solution, people look at applications that build on top of that data. One such type of added-value applications on the tourism sector is based on automatically generating tourist plans. Currently, there are some platforms that provide related services. For instance, Tripomatic[4] is a powerful tool for travel planning, but requires the user to select places to visit and manually set up the plan. As another example, CityTripPlanner[5] is able to automatically generate tourist plans, but it does not suggest places to eat in a reasonable way according to user pace, hunger and restaurant timetables.

In this paper we present PlanTour, a new tool that uses an automated planning approach to generate tourist plans. This planning system was built for the ONDROAD project, a framework for the management and planning of digital contents and services provided for bus travelers of the ALSA[6] company. Within this project, PlanTour is the sub-system in charge of building the tourist plans for users visiting a particular city or region.

In terms of planning applications, the main contributions of this work can be summarized as:

- The automatic composition of the initial state and goals using information from a social network. This partially tackles the problem of the slow start of similar systems, which have to wait until a sufficient amount of data is collected to run properly.
- The modeling of user drives as part of the planning process, to obtain more realistic plans in terms of deliberative reasoning. Specifically, suggested restaurants are smoothly integrated in tourist plans just when it is expected that the user is hungry based on her preferences.
- Modeling the problem of recommending tourist POIs as an oversubscription planning task, given that the available alternatives (visiting POIs)

[4] www.tripomatic.com
[5] citytripplanner.com
[6] http://www.alsa.es/en

are many more than the ones a tourist can carry out with her time and budget.

- Successful application of compiling away soft goals using the approach by Keyder and Geffner [8] to solve the oversubscription planning task.
- The domain-dependent problem decomposition using a clustering algorithm. Resulting sub-problems match the problem of finding a plan for a single day where candidate POIs are geographically close.

The following sections describe the problem formulation, the architecture with all its components, the representation of the domain, the experimental results, the related work, and finally, conclusions and future work.

2. Problem Formulation

The generation of personalized tourist plans has been previously proposed as a *Tourist Trip Design Problem* (TTDP) [9]. This generic class of problems comprises a set of candidate POIs together with their associated attributes (i.e., type, location, timetable, etc.), travel time between POIs, user-dependent functions relative to POIs (i.e., satisfaction, expected duration, etc.), the trip time-span and the daily time limit. A quality solution to a TTDP is expected to suggest a daily plan that respects the constraints imposed by POIs properties while maximizing the user utility. Modeling a concrete TTDP will depend on the available tourist data and the user inputs. In PlanTour the TTDP is modeled with the following elements:

- A fully-connected graph (V, E) where every node $v \in V$ represents a POI and every edge $e \in E$ represents a path between two different locations. A function $d_t : E \to \mathbb{R}^+$ represents the time it takes to traverse an edge. The possible routes between the locations do not always use the same transportation means. Depending on the distance between two locations, the edge can be traversed either walking or by car. Thus, the values of the time function d_t are computed considering these two options.

- A tuple of properties associated to every POI in V. Thus, each property is denoted by its function name (e.g *type*(v), *duration*(v)). These properties are summarized in Table 1.

Table 1: Properties included in the TTDP formulation to characterize POIs.

Property	**Values**
Type	nature, building, sport, eating, events, ...
Subtype	subtype within a type (e.g. a museum is a subtype of building)
Score	the POI *total_score* taken from MINUBE (POI popularity)
Timetable	the opening (*t_open*) and closing time (*t_close*)
Duration	estimated time to visit in function of the subtype (t_p)
Location	the POI geo-coordinates (latitude and longitude)

- A tuple of properties associated to the user for encoding her constraints and preferences. For instance, the time-span *[init_trip,end_trip]* for the whole visit. Table 2 shows the list of user properties. Given that our plans also include eating actions, we define $V_r \subset V$ as a special set of POIs representing the places where the user can eat. Plans should consider the number of times the user wants to eat in a day (*times_eat*). Also, the selection of restaurants should respect the minimum (h_{min}) and maximum (h_{min}) time that the user considers reasonable to wait until eating again. These constraints forbid the concatenation of two POIs belonging to V_r, but forces to include some of them when the user is expected to be hungry.

A solution to this TTDP is a sequence of time-stamped actions, which includes tourism activities and the necessary travels to perform these activities. This plan should satisfy the time-span constraint for the whole visit and the daily time constraints, where the user wants to perform the activities.

3. System Architecture

The PLANTOUR architecture is composed of three main sub-services (see Figure 1). The Tourist Plan Manager (TPM) receives the inputs for the PLAN-

Table 2: Properties of users that can be specified from his/her preferences.

Property	Description
location	where the user is
[init_trip,end_trip]	the trip time-span
h_{min}	minimum time to pass to recommend a place to eat
h_{max}	maximum time to pass to recommend a place to eat
times_eat	maximum number of times the user eats on a day
init_time	the first time the user is available in the day
end_time	the last time the user is available in the day. When this time is due, we assume the user is tired to do more things

Tour planner. The inputs of PlanTour are: the city or region the user is going to visit; when s/he is going to be available (time to arrive and leave the place); and possibly, some constraints and preferences. Users are not required to input a complete list of their constraints and preferences. Therefore, we provide default values for these properties. The default values are: *[init_time, end_time]* = [10:00 AM, 8:00 PM], the time interval that the user can use to perform an activity, *times_eat=2*, only two POIs from V_r will be recommended in a day; and $h_{min} = 2$ and $h_{min} = 5$, so that at least two and at most five hours should pass before recommending a place from V_r again.

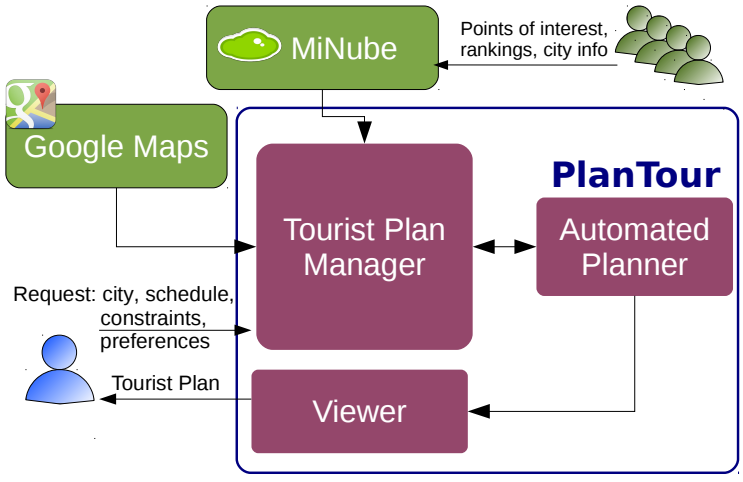

Figure 1: Architecture of PlanTour.

When a plan is requested by the user, the TPM retrieves the needed information from the MINUBE social network. This information is pre-processed to create planning problems in the standard declarative language used by the automated planning community, PDDL (*Planning Domain Definition Language*) [10]. The Automated Planner module solves the planning tasks using two different metrics: a) maximizing the user utility of visiting places; and b) minimizing the total traveled route. The Viewer module collects the final plans and returns a web page where all locations appear together in a map provided by Google Maps API. Additionally, it shows a map per day, and each location in an individual map. In the following subsections we describe in more detail each module.

3.1. MINUBE *Service*

MINUBE is a traveling social network where users can: get inspiration from other users to decide the destination of their next trip; create a selection of POIs for the desired city or region; and share with others their experience, photos and recommendations.

In PLANTOUR, interesting POIs and their properties are retrieved from MINUBE to formulate the planning problem. This social network maintains a data base created by crowdsourcing where the minimal piece of information is the POI. Users continuously add and update POIs with photos, opinions, recommendations, etc. These POIs are reviewed by a group of experts in MINUBE that provide quality assurance over the basic POI information: location; phone contact; and web site. Specifically, we retrieve from MINUBE the type, subtype, geo-localization and user votes used to compute the score. Expected duration and timetables are frequently not available. We provide a default value per POI type, suggested by domain experts. We could easily replace the queries to this service for any other similar service such as Tripadvisor.

3.2. Google Maps

Google[7] provides a map service with a highly visual interface portable on all platforms and devices. This service has detailed street information together with distance and time estimations between locations. It includes an open API allowing customization of the map output. We use Google Maps in two different ways. In the first one we query the service to collect estimated time to move among the main locations in several cities and zones. In the second one, we use the service as part of the plan visualization. Again, we could replace this service for similar ones, as OpenStreetMaps.[8]

3.3. Tourist Plan Manager

This module handles TTDPs in PLANTOUR, generating planning tasks that are given to an automated planner, using the information extracted from MINUBE services. For most regions or cities the list of POIs can be huge, and there will be no plan that can visit all of them. Therefore, in the first step, the TPM queries MINUBE to retrieve the set of the best N POIs for the selected destination. The system can use any value of N, but an obvious constraint is to set N to $N \geq k \times n$, where k is the number of days and n is the maximum number of POIs that can be visited per day. We suggest n to have values between 20 and 50 to include enough diversity of places in the routes.

The set of best POIs is built from a ranking computed as the average between the popularity of POIs and the aggregated score each POI has received from the user's contacts in the MINUBE network. In case the user has no contacts, the ranking is only determined by the general popularity of each POI. The POIs' popularity is based on the number of entries, opinions, photos and videos. Thus, this process is an initial filter that generates a subset of POIs which are considered the best ones by the user contacts and the public in general. In addition, we run a second filter that removes all places that are not currently

[7] http://www.google.com/maps/
[8] http://www.openstreetmap.org/

available. For instance, POIs with type (temporal) *event* that are not being held in the visit interval.

The number of possible plans is exponential in N. Thus, if N is still big enough to include all potentially interesting POIs, no automated planner is able to handle planning tasks of this size. Therefore, the second step consists of splitting the TTDP into single-day problems, assigning to each sub-problem a disjoint subset of POIs that are geographically close. This step tries to mimic the way in which people prepare their tourist visits in plans for several days. The solution to the whole problem is computed as the concatenation of the solution to each sub-problem.

When clustering the POIs, we first tried the k-means algorithm [11], using the (x, y) coordinates of the geo-localization as features, and setting the number of clusters to the number of available days. However, this algorithm presents some problems with cities or regions where their POIs are not evenly distributed: plenty of POIs can be grouped together while few others can be far away. Then, we decided to implement a balanced k-means [12], a modification to the original algorithm that imposes the constraint of having the same number of elements in each cluster. The algorithm works as follows: an initialization step randomly selects a centroid for each cluster from the set of POIs. Then, an iterative procedure computes these steps:

- arrange the distances from each POI to each cluster centroid in ascending order;
- following this ordering, assign a POI to a cluster if it has not been assigned yet, and if the cluster has not exceeded the limit of N/k POIs; and
- using the previous assignment, re-compute the cluster centroids as the mean of the elements in the cluster.

The output of the balanced k-means can be seen as a way of giving a balanced pace of visits for the whole trip. Figure 2 shows the resulting balanced clusters for three different days in London after running the algorithm.

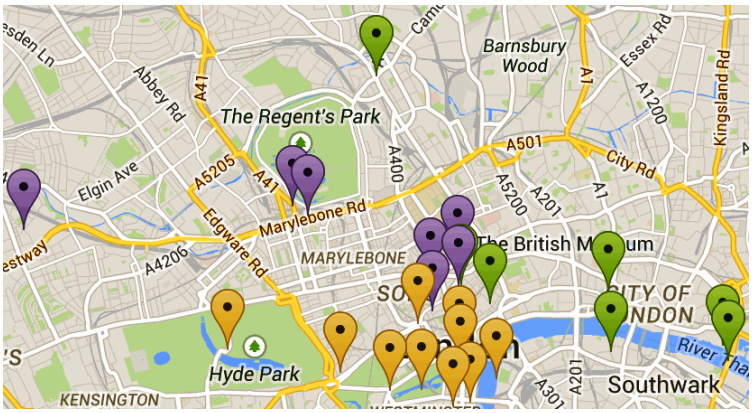

Figure 2: A balanced distribution of clusters for three different days in London.

Additionally, the TPM computes the estimated time for driving and walking from each POI to the rest of POIs in the same resulting cluster. We only use the best way to move in the planning problem: if it is a short distance, the provided route suggests walking; otherwise, it suggests driving. Figure 2 shows that most POIs are very close together, so the user could walk. However, there are some POIs with a large distance to the rest, which suggests the use of a car or public transport.

Time estimations are computed with the POI geo-coordinates using two different procedures:

1. Google Distance API[9] that provides estimated travel time between two locations. This service has restricted free use per day and per computer. Therefore, we use it below the given bound, obtaining the estimated time starting from the most popular POIs, since they are likely to appear in most plans. The time estimations that can not be retrieved with this API are computed with the second procedure.
2. Computing the estimated time dividing the distance between two given POIs by the average velocity of the traveling mode. The distance is the result of the Haversine formula [13], which computes the shortest distance

[9] `http://code.google.com/apis/maps/documentation/`

between two points in the surface of the Earth using geo-localizations. The average velocity is set to 4 km per hour if the distance is short (walking will be suggested) or 50 km per hour if it is a long distance. These speeds are similar to the ones used by Google, and in the case of the car it is the average of the speed limit in towns in Europe. These time estimations are less accurate than Google estimations given that in this case the real paths and road directions are not being considered.

The relevant information for formulating a per day sub-problem of TTDP is (1) the sub-graph of POIs being considered this day, (2) the properties of these POIs, and (3) the user properties and constraints. This information for a given day is compiled into a planning task, which is formulated in PDDL. The TPM generates a problem per day and invokes the automated planner in two iterations. The first one solves a utility maximization problem with the objective of providing the user a plan for visiting the POIs that give her the maximum satisfaction. The second one solves a distance optimization problem; once the set of POIs have been decided, the user can spend the minimum time going from one POI to another. Finally, the TPM combines the single-day plans returned by the automated planner into a single tourist plan. Figure 3 shows an example of a returned plan.

Tourist Plan
10:45 visit big-ben [60.00]
11:45 walk big-ben st-james-park [20.00]
12:05 visit st-james-park [30.00]
12:35 free-time [120.00]
14:35 walk st-james-park little-frankies [15.00]
14:50 eat little-frankies [60.00]
15:50 walk little-frankies picadilly-circus [20.00]
16:10 visit picadilly-circus [15.00]

Figure 3: A plan for a single day. Each row shows the time of the day, the activity to perform and the estimated duration (in minutes).

3.4. Automated Planner

Automated Planning is a subfield of AI that develops problem solvers that find solutions to problems in the form of sequences of actions, called plans [14]. These plans should take a system (or human) from a given initial state to a state where a set of goals are achieved. Most planners are domain-independent, so the code of the planner does not include any knowledge on the domain model. Therefore, they take as input a planning task and their output is a plan. A planning task is usually represented declaratively with (1) a domain-model, a specification of which actions can be applied in the environment, and (2) a problem, a description of the initial state and goals and, optionally, a metric to be used by the planner. If given, the metric will tell the planner that the user wants the plans to minimize (maximize) a given criteria, such as time, cost or satisfaction. Each action is modeled similarly to a rule, and includes a set of preconditions that must hold in a state in order the action to be applicable, and a set of effects that are expected to change the state once the action is applied. The effects can also contain information on the cost of applying the action according to different metric criteria. The cost of a plan according to a given metric is computed as the sum of the costs of the actions in the plan according to that criteria.

One of the main benefits of using automated planners is that knowledge engineers can focus on modeling the application requirements and then they can use an off-the-shelf solver that has been developed and optimized by the planning community. In our case we model TTDPs in PDDL, the standard language in the community. And we use the Metric-FF planner [15], since it is one of the very few planners that can handle all the language features we need for our system, such as numeric preconditions in the actions descriptions and metric optimization. We do not need other kinds of more complex planning systems as temporal planners (they can reason about concurrent temporally annotated actions), given that our planning tasks do not have concurrent actions; a user can only perform an action at a time. Metric-FF is a heuristic search planner. Starting in the initial state, the planner generates a search tree where

nodes represent states and arcs represent actions. METRIC-FF uses two search algorithms, Enforced generate Hill-Climbing (EHC) and Weighted Best-First Search, and a heuristic function that estimates the cost of a plan from each state to a state where the goals are true. The reader is referred to its papers for more information on how it works [15]. We have adapted the planner to skip EHC and run the WBFS directly. Given a planning task, the TPM calls the planner twice. The first call to the planner solves a problem with the metric of maximizing the user utility. The second call solves a problem with the metric of minimizing the distance to travel among POIs. Since, at this point, the system already has a feasible solution provided by the first call to the planner, we give the planner a short time bound (we are using 10 seconds) to return a solution as fast as possible. The goal of this second optimization step is to obtain shorter routes under the assumption that users prefer to have some free time in their plans rather than having a tight schedule.

3.5. Viewer

This module creates a web page that includes external information from Google maps. Figure 4 shows an example of the resulting web page and its links. The maps included in the final version are of four different types:

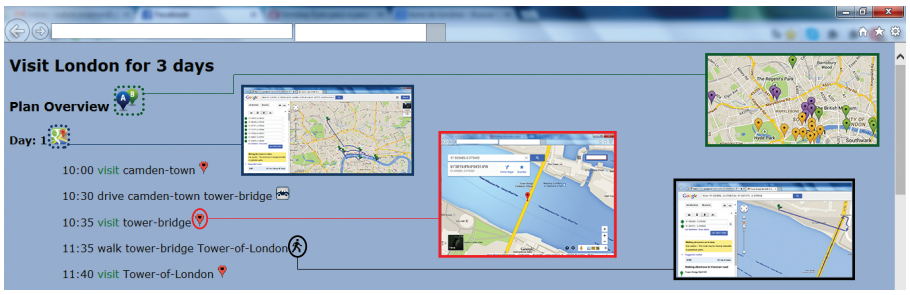

Figure 4: Image of the final web page, with the tourist plan and the links to the city of London in Google Maps.

1. Plan Overview: a global map with all the POIs locations included in the plans of all days.

2. Map per day: map that only includes the locations of the day with the suggested route.
3. Location point: map where there is only one location (a POI).
4. Movement between two points: the route between every two points in the plan.

4. PDDL Representation

In this section we describe how a TTDP for PlanTour is modeled using PDDL. Planning tasks are modeled with PDDL in two files: the domain and the problem file. The domain contains a high-level declarative description of a transition system in terms of object types, predicates and actions that transform states into other states. The domain file is usually the same for all planning tasks within an application domain. The problem file contains the description of a particular situation. It defines the set of objects participating in the planning task, an initial state, a set of goals and, optionally, a metric. In our application, a planning task corresponds to a sub-problem for a single day, which includes all user properties and POIs assigned by the clustering algorithm.

We have defined two different domain models for PlanTour. The *utility model* focuses on filtering the candidate POIs to obtain a feasible subset that maximizes the user utility while respecting the constraints imposed by the need to travel among locations. The *travel model* optimizes user travels among POIs in order to offer some free time. We first describe the common parts of both models and then their differences.

First, a hierarchy of types is defined in the domain files for POIs. We use the retrieved types and subtypes from the database of minube. POIs are then defined as objects of their corresponding type in the problem file. The user properties and the rest of POIs properties are defined as numeric fluents. Additionally, a numeric fluent *current_time* is used to simulate the passing of time, and the numeric fluent *hunger* is used to keep track of the corresponding user's drive. We have defined the following actions:

- **visit**(u, p) requires that the user u is at the same location as the POI p. Also, the user must have enough time to visit the place, the place has to be open until u finishes the visit, and u should not have visited p before. Another condition is that u is not very hungry. This is controlled by keeping track of the elapsed time from the last visit to a POI $v \in V_r$ and checking that it does not surpass a given threshold (i.e., $hunger \leq h_{max}$). The effects of this action are that u has visited p, the *current_time* is increased by the duration of the visit, the user utility is increased as a function of the score (rating) of the POI and *hunger* is increased too. The complete description is shown in Figure 5.

```
(:action VISIT
 :parameters (?u - user ?p - poi ?d - day)
 :precondition (and (user-at ?u ?p)
                    (<= (hunger) (h-max))
                    (>= (current-time ?u ?d) (open ?p ?d))
                    (<= (init-time-available ?u ?d) (current-time ?u ?d))
                    (<= (+ (visit-duration ?p) (current-time ?u ?d))
                        (close ?p ?d))
                    (<= (+ (visit-duration ?p) (current-time ?u ?d))
                        (end-time-available ?u ?d))
                    (normal-mode)
                    (not (visited ?u ?p)))
 :effect (and (visited ?u ?p)
              (can-walk)
              (increase (total-score) (score ?p))
              (increase (current-time ?u ?d) (visit-duration ?p))
              (increase (hunger) (visit-duration ?p))))
```

Figure 5: The PDDL representation of the **visit** action.

- **walk**(u, $place_1$, $place_2$) refers to the walking movement of the user u from location $place_1$ to $place_2$. It is only possible if the POIs are close in the map (the journey duration is less than 1.5 hours). The user should be at $place_1$ and at the end the user will be at $place_2$. We control that the plan does not include two contiguous movements, since we assume the graph is fully connected. Therefore, this action can only be executed if the previous action was not a move (walk or drive) action. As effects, the *current_time* is increased by the time it takes to walk between $place_1$ and $place_2$, and *hunger* is also increased as in the previous action.

- **drive**(u, $place_1$, $place_2$) is similar to the previous one, but using the car. The main difference is that the POIs have to be far from each other.

- **eat(**u**,** *eat_place***)** refers to the action of the user u going to a restaurant *eat_place*. The user should be at *eat_place*. Also, the number of $v \in V_r$ that u has visited should not exceed *times_eat*, u's *hunger* has to be between the given thresholds, $h_{min} \leq hunger \leq h_{max}$, the user should have enough time to go to the restaurant, and the place has to be open until u visits the place. The effects of this action decrease the user's *hunger*, increase the *current_time* by the duration of the visit, increase the user's utility by the place score, and add that the user has visited the place *eat_place*.

- **pass-time(**u**,** *time-slot***)** refers to the action of u doing nothing (as a no-op). u should have enough time to execute this action during a *time-slot* duration. The effects of this action are that *current_time* passes the corresponding *time-slot*, the user's *hunger* increases proportional to the value of *time-slot*, and the user's utility decreases proportionally to the wasted time. This action is essential in the domain model because there are some cases where performing any other action is not possible. Mainly, when their preconditions are not true at the current time, but they become true later after some time (e.g. some place opens).

In the problem files, the initial state includes the user's properties (Table 2) and the information from the formulated TTDP regarding a particular day. Thus, the set of POIs assigned by the clustering algorithm determines the set of facts belonging to the initial state; that is, the properties and the complete distance graph for this set of POIs. The goals depend on the model being used. Now, we will describe the domain and problem differences for the two planning steps.

4.1. Utility Model

The objective of this model is to maximize the *goal_score* for each problem:

$$goal\_score = \sum_{i=1}^{|V|} score(v_i) \times visited(v_i)$$

where $|V|$ is the number of candidate POIs for the day and the function *visited*: $V \rightarrow \{0, 1\}$ represents whether each $v_i \in V$ is included or not in the final plan. Since not all POIs will be visited, instead of defining visiting them as hard goals, they are considered as soft goals. Hard goals must be met by any valid plan. Soft goals are not forced to be achieved by the plan. However, not achieving them penalizes the quality of the plan. When planning for the utility model, all goals are declared as soft goals. The goal set is defined as visiting all candidates POIs. We handle soft goals following the approach described in [8], where all goals are hard goals (i.e., they must be true at the end of the plan), but each domain action achieving a goal has a skip action achieving the same goal but with a high penalty. In our case, during the "normal-mode" the user visits as many POIs as she can until time runs out. Then, the "end-mode" achieves the rest of goals actually skipping the visits. Using this model, the planner has to achieve all goals, but only the plan during the "normal mode" is used as the tourist plan.

4.2. Travel Model

In this model the metric consists of optimizing the total traveled distance:

$$goal\_score = \sum_{n=1}^{|G|-1} \text{distance}(p_n, p_{n+1})$$

where $G \subseteq V$ is the set of goals (POIs) included in the tourist plan generated with the utility planning task. In this new task all goals are hard. Therefore, the goals not belonging to G are not included in the new set of goals, and the domain file does not include skip actions for solving a goal. This modification is safe because the solution to the utility planning task guarantees the solvability of this new problem. Indeed, the traveled distance in the plan for the utility planning task works as an upper bound of the cost of the plan in the second model, and new solutions are used just in case they have better quality.

5. Experimental Results

The ALSA company wanted the first prototype to work for two different regions in Spain: Granada and Asturias. But, in order to test the behavior of the system, we include here results for a group of other cities and regions (geographic zones) around Europe, given that most users of MINUBE come from Europe. We have generated common scenarios in four cities and three regions. We selected Madrid, Oviedo (Asturias), London and Rome as cities; and Granada, Belgium and Hesse (as regions/countries). We have generated scenarios varying the number of the days in the travel from two to seven.

We have evaluated the results considering the run time, including pre-processing and total planning time, and the plan quality, measured as the sum of scores of the visited POIs. Additionally, we have evaluated the effect of using the *hunger* drive as part of the decision making. The experiments were run on a 2.3Ghz Intel i5 processor with 4GB of RAM. We developed TPM in Python. The automated planner we used, Metric-FF [15] was developed in C. The whole system runs on Ubuntu 14.04 LTS.

Table 3 shows the results in terms of run time, where t_g is the time for generating all planning tasks and t_t is the total run time, including the generation time. The generation time corresponds to the CPU time spent in pre-processing information from MINUBE, performing the clustering of POIs and writing the planning task in PDDL. As it can be seen, t_g never exceeds five seconds and in most cases it represents only a small fraction of the total time. On the other hand, t_t aggregates t_g and the time spent in planning a route for each day. Given that our approach performs a decomposition strategy and each day has a limited set of pre-selected POIs, the total time only increases linearly in terms of the number of days. The average of t_t per day is around 3.81 seconds. This execution time is acceptable for a route computation if we take into account the time it takes humans to generate routes.

Table 4 shows the number of POIs included per experiment and the number of POIs visited in the generated solutions. This a proportion of solved goals

Table 3: Time to generate the PDDL planning task (t_g) and the total run time (t_t) in seconds for different cities and regions.

Cities		**Days** 2	3	4	5	6	7
Madrid	t_g	0.83	1.47	1.36	1.62	2.26	2.40
	t_t	5.73	7.32	7.43	11.22	11.38	37.01
Oviedo	t_g	0.66	0.87	1.18	1.48	2.78	3.46
	t_t	12.66	23.47	10.50	23.37	28.56	34.56
London	t_g	2.50	2.65	3.34	2.95	2.95	4.70
	t_t	4.21	5.01	5.99	16.18	13.14	18.83
Rome	t_g	2.19	1.81	2.80	2.25	2.39	4.65
	t_t	15.29	15.05	18.84	27.68	27.44	35.51
Granada	t_g	0.75	1.05	1.95	1.54	2.89	1.94
	t_t	12.79	13.65	19.93	17.14	24.01	20.97
Belgium	t_g	0.58	0.88	1.04	1.26	1.36	1.63
	t_t	12.47	3.74	13.77	26.78	32.50	34.13
Hesse	t_g	0.53	0.50	0.50	0.44	1.43	1.91
	t_t	4.79	16.48	3.34	6.97	6.97	15.66

with the oversubscription planning task with respect to all the included POIs in the utility model design. For these experiments, we use the balanced k-means to have a similar number of POIs per problem. Previous experiments showed that the standard k-means produced very unbalanced daily plans. Thus, the generated solutions were not usable from an application point of view. Therefore, no additional tests were performed using that approach. The ratio between POIs included and POIs visited is between 16% (London) and 34.55% (Hesse). This ratio shows that the planner has a high number of alternatives and in all cases more than 50% of the candidate POIs are discarded in the first iteration of the planner.

Table 5 shows the quality of the generated solutions. We include two different metrics, q and q_{best}, where:

- q is the ratio between the quality of a plan (sum of scores of visited POIs shown as the numbers on the left in cells of Table 4) and the total sum of scores from all POIs included in the problem (shown as the numbers on the right in cells of Table 4).

Table 4: Number of POIs visited in a k-days trip (left number in each cell) vs. number of considered POIs per experiment (right). The maximum number of POIs is $40k$, where k is the number of days. The average number of visited POIs per problem is between 9 and 10.

	Number of days (k)					
Cities	2	3	4	5	6	7
Madrid	20/78	29/114	41/150	52/190	66/264	54/264
Oviedo	25/71	36/99	46/130	60/164	67/190	68/217
London	12/ 77	24/111	24/149	36/181	48/215	42/253
Rome	21/80	33/119	41/157	54/192	63/225	75/257
Granada	22/69	34/101	45/129	47/153	53/179	54/198
Belgium	23/75	28/112	23/148	46/184	55/218	70/250
Hesse	23/73	32/86	32/89	35/91	54/126	57/165

- q_{best} is the ratio between the quality of a plan with N visited POIs and the sum of the best N POIs available in the problem.

The ratio q measures the resulting quality with respect to the maximum quality that can be obtained if all POIs in the city/region could be visited. However, given that it is not feasible to visit all POIs (as we already have seen in the previous table), we included metric q_{best} to consider the situation in which the N best POIs are naively selected in each problem, where N is number of visited POIs in each plan. We compute the sum of scores of these N best POIs and compare it to the quality obtained by our plans. These results show that q is around a 50% of the total score while q_{best} increases to around 70%-80%. High values of q_{best} indicate the planner is able to schedule in a day the most relevant POIs for the cluster. However, constraints on user movements and eating times prevent obtaining higher ratios.

Regarding the *hunger* drive, we wanted to see the difference between the number of meals in the generated plans (m_h) and the number of meals if we naively select the best N POIs from the set of available POIs for each day (m_N), where N is number of visited POIs in each plan. Table 6 shows the maximum, the minimum and the mode of the number of recommended POIs to eat in each city per scenario. Variations of m_N depend on POI relative scores compared to the rest of POIs in the same cluster. If there are many restaurants with a good

Table 5: Ratios of the plan quality. Row q_{best} is the ratio between the quality of the obtained plans and the utility of visiting the best N POIs of each day. Row q is the ratio between the quality of the obtained plans and the sum of the utilities of all the retrieved POIs.

Cities		**Days** 2	3	4	5	6	7
Madrid	q_{best}	0.94	0.91	0.89	0.82	0.85	0.81
	q	0.59	0.57	0.58	0.70	0.61	0.41
Oviedo	q_{best}	0.90	0.88	0.90	0.90	0.84	0.77
	q	0.88	0.85	0.81	0.91	0.81	0.77
London	q_{best}	0.78	0.92	0.94	0.90	0.92	0.46
	q	0.51	0.76	0.63	0.75	0.80	0.37
Rome	q_{best}	0.90	0.88	0.92	0.94	0.74	0.91
	q	0.87	0.87	0.87	0.90	0.86	0.88
Granada	q_{best}	0.95	0.86	0.82	0.67	0.93	0.88
	q	0.93	0.87	0.84	0.83	0.88	0.74
Belgium	q_{best}	0.89	0.77	0.82	0.80	0.81	0.81
	q	0.89	0.71	0.55	0.72	0.70	0.75
Hesse	q_{best}	0.88	0.92	0.86	0.78	0.95	0.87
	q	0.85	0.84	0.62	0.54	0.81	0.81

score (e.g., as in some clusters in Madrid), a naive selection of POIs will end up with unrealistic plans. Even though a number of restaurants could be filtered out in each cluster, our planning approach is able to consider all of them and select the suitable ones in terms of time and planned route. Nevertheless, if there is no eating POI in a cluster, the system cannot recommend any place to eat. For instance, in some cases in London and Belgium, there are no recommendations on some days (minimum 0) because no restaurant passes the initial filter of POIs around the cluster area.

The *hunger* drive is useful to recommend meals when it is expected that the user wants to eat. The planner can propose a restaurant considering the time interval that has passed without the user eating, which should be in the interval $[h_{min}, h_{max}]$. Our default values $[2, 5]$ are flexible enough for generating plans with 2 restaurant recommendations in most of the evaluated scenarios. Conversely, the naive selection shows that unrealistic routes can be obtained if eating places are not differentiated from other non-eating potential POIs.

Table 6: Maximum, minimum and mode for the number of recommended POIs to eat (meals) when using the *hunger* drive (m_h) in our approximation or when selecting best POIs (m_N) naively.

Cities		*max*	*min*	**Days** 2	3	4	5	6	7
Madrid	m_h	2	1	2	2	2	2	2	2
	m_N	7	0	1	0	4	3	4	0
Oviedo	m_h	2	1	2	2	2	2	2	2
	m_N	7	0	3	2	2	5	4	5
London	m_h	1	0	1	1	1	1	1	1
	m_N	3	0	0	0	0	0	1	0
Rome	m_h	2	1	2	2	2	2	2	2
	m_N	3	0	0	1	2	2	0	1
Granada	m_h	2	1	2	2	2	2	2	2
	m_N	6	0	3	3	1	4	2	1
Belgium	m_h	2	0	1	1	2	2	2	1
	m_N	3	0	0	0	0	0	0	0
Hesse	m_h	2	1	2	2	1	2	1	2
	m_N	4	0	4	2	2	1	1	1

6. Related work

The development of applications for building customized tourist plans has increased in recent years due to the popularization of mobile devices and the continued growth of tourism services. Research efforts on solving TTDP problems varies in many aspects, regarding the acquisition of user preferences, the algorithmic approach for filtering interesting POIs and the technique for arranging these POIs in tourist routes [9]. The generation of these tourist routes is commonly solved by some version of the *Team Orienteering Problem* (TOP). The problems are posed as optimization tasks and solved using diverse techniques such as classical optimization methods or stochastic local search. The main benefit of using automated planning consists of being able to easily include constraints into the models. For instance, we have included some of the features that are listed as future work in the review paper [9], such as including free time, or flexibly integrating restaurant suggestions by using drives (as hunger).

CityTrip Planner [4] creates tourist guides based on a small questionnaire

that tries to obtain the user preferences and constraints. It was initially designed for five cities in Belgium. The POI database is populated and maintained up to date by the tourist office of the different cities. It has continued to grow, and now it includes several other cities around the world. They model the problem as an extension of TOP with time windows, and solve them using stochastic local search. The generated solution can be downloaded to a mobile GPS device allowing the tourist to track her route. However, as other TOP approaches, they do not suggest free time for the user and they do not base their recommendations on user's drives.

Automated Planning has been previously used in some tourism-related application. SAMAP [2], captured and updated a user model according to previous user behavior when she visited other cities. The list of attractions is selected using a Case Based Reasoning approach, which retrieves similar users that have already visited the same city. The planning model allows the system to compute plans containing POIs that are more likely to interest the user, together with directions on how to get from one place to another. Our approach follows the same ideas, but addresses its main drawback, the scarcity of initial tourist information. In SAMAP, they had to manually provide all the information, while we access it directly from a social network. Also, we use drivers for flexibly including eating actions, and use external services for map related computations (in SAMAP, maps were manually defined in a file). eTourism [16] considers eating actions, but they are modeled as dummy actions that add time gaps into the plan while the user stays in the last visited POI, so she can use this free time to eat. In PlanTour, eating actions correspond to real recommendations and the time to go to restaurants is also taken into account. In addition, eTourism also has the problem of having very little initial information, as it was only deployed for the city of Valencia (Spain). Le Berre et al. [17] proposed a planning model for building a personalized museum visit. Their model is similar to the PlanTour model in the way they encode current-time, or POI properties. In their case, properties are utility and expected duration for viewing an artwork. This system also suffers from the scarcity of available information, specially because

topology of museums (i.e., graph for rooms distribution and artwork locations) as well as artwork relevance are not frequently available and need to be fetched manually.

In PlanTour we have used existing web services to enrich our application capabilities. This is also the case of *Itinerary Planning* [18], which estimates distances between POIs using Google API and retrieves POIs utility with the average score from Yahoo Travel. However, the user has to provide the initial list of POIs, and the system needs a MapReduce framework as they try to explore the space of all possible itineraries.

TripBuilder [19, 20] is a system based on Flickr and the itineraries followed by different tourists. The POI information is extracted from Wikipedia. They model a problem that explicitly needs a source POI, a destination POI, the total number of POIs to be visited and an optional set of POIs not to be visited. The method is an instantiation of the Generalized Maximum Coverage, extracting the routes from the timestamp in the published photos in Flickr. In contrast, we have POIs with popularity scores and no connection among them. Our system can generate routes that were not predefined by other users, thus we could propose more innovative solutions. Besides, TripBuilder does not recommend any meals or includes free time in its routes. Another system based on Flickr [21] creates routes between 2 to 6 hours within 6 different cities in China. In this case, routes are extracted with the geo-spatial meta-data of Flickr pictures and the information from Google places. This system is restricted to plans of a few hours and it does not include particular recommendations for meals or free time.

7. Conclusions and Future Work

We have presented PlanTour, a new sightseeing recommender system that is able to work properly for most popular tourist destinations thanks to its ability of automatically gathering tourist information directly from a social network. The system takes into account the number of days for the visit and plans accordingly. It divides the city/region to visit by the number of days and provides

a tourist plan for each day that includes highly valued POIs by MINUBE users, as well as the routes between these POIs. These plans are aware of the user drive of being hungry, as they suggest good places to eat when it is more likely to satisfy user's needs.

Among the main strengths of PLANTOUR we can mention: use of a declarative model of the task that allows easy modeling and maintenance; explicit consideration of relevant drives (only hunger right now); integration of human provided knowledge in a social network with an efficient domain-independent problem solver to solve a hard combinatorial task; or two-steps optimization task to improve users' utility (satisfaction) and time to spend visiting a place as shown in the experimental results. Some current weaknesses of PLANTOUR are: the current dependency on MINUBE information, though we already are obtaining information from Yelp and Google Places; the limit on accesses to Google Maps, though we can already obtain similar services from OpenStreetMaps; there is no on-line adaptation to the users; or lack of reasoning on some other relevant drives, as fatigue.

Having modeled both users preferences and drives within the PDDL planning tasks facilitates the inclusion of new features to the decision making process. For example, in the future we want to modify the model to include fatigue as a drive that relates the user stamina with the number of continuous visits or the speed of visits. In case the fatigue is estimated to fall down below a given threshold, the system would suggest POIs where the user could rest, such as parks or coffee shops. Using a declarative model, such as the one in PDDL, provides knowledge engineering advantages over traditional techniques that pose the problem as a classical optimization problem. Additionally, users can manually provide plans in MINUBE. So, in the future, we would like to reuse these plans to generate better plan recommendations. Also, following other works in interleaving planning, execution and learning [22], we would like to generate an integrated tool that could help the user while executing the tourist plans to re-plan on-line, repairing the plan under execution and learning user models that improve future planning episodes.

Acknowledgments

This work was supported by the Spanish project ONDROAD (TSI-090302-2011-6), MICINN project TIN2011-27652-C03-02 and MINECO project TIN2014-55637-C2-1-R.

References

[1] T. Berka, M. Plößnig, Designing recommender systems for tourism, in: Proceedings of the 11th International Conference on Information Technology in Travel & Tourism, 2004.

[2] L. Castillo, E. Armengol, E. Onaindía, L. Sebastiá, J. González-Boticario, A. Rodríguez, S. Fernández, J. D. Arias, D. Borrajo, SAMAP. A user-oriented adaptive system for planning tourist visits, Expert Systems with Applications 34 (2).

[3] A. Moreno, A. Valls, D. Isern, L. Marin, J. Borràs, Sigtur/e-destination: ontology-based personalized recommendation of tourism and leisure activities, Engineering Applications of Artificial Intelligence 26 (1) (2013) 633–651.

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

[5] M. Rodriguez-Sanchez, J. Martinez-Romo, S. Borromeo, J. Hernandez-Tamames, Gat: Platform for automatic context-aware mobile services for m-tourism, Expert Systems with Applications (2013) 4154–4163.

[6] L. Manikonda, T. Chakraborti, S. De, K. Talamadupula, S. Kambhampati, AI-MIX: using automated planning to steer human workers towards better crowdsourced plans, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada., 2014, pp. 3004–3009.

[7] J. P. Lucas, N. Luz, M. N. Moreno, R. Anacleto, A. A. Figueiredo, C. Martins, A hybrid recommendation approach for a tourism system, Expert Systems with Applications (2012) 3532–3550.

[8] E. Keyder, H. Geffner, Soft goals can be compiled away, Journal of Artificial Intelligence Research 36 (1) (2009) 547–556.

[9] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, A survey on algorithmic approaches for solving tourist trip design problems, Journal of Heuristics 20 (3) (2014) 291–328.

[10] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, Journal of Artificial Intelligence Research (2003) 61–124.

[11] S. P. Lloyd, Least squares quantization in *pcm*, unpublished Bell Laboratories Technical Note. Portions presented at the Institute of Mathematical Statistics Meeting Atlantic City, New Jersey, September 1957. Published in the March 1982 special issue on quantization of the IEEE Transactions on Information Theory (1957).

[12] S. Zhong, J. Ghosh, A unified framework for model-based clustering, The Journal of Machine Learning Research 4 (2003) 1001–1037.

[13] C. Robusto, The cosine-haversine formula, American Mathematical Monthly (1957) 38–40.

[14] M. Ghallab, D. Nau, P. Traverso, Automated planning: theory & practice, Access Online via Elsevier, 2004.

[15] J. Hoffmann, The METRIC-FF planning system: Translating "ignoring delete lists" to numeric state variables, Journal of Artificial Intelligence Research 20 (2003) 291–341.

[16] L. Sebastia, I. Garcia, E. Onaindia, C. Guzman, e-tourism: A tourist recommendation and planning application, International Journal on Artificial Intelligence Tools 18 (5) (2009) 717–738.

[17] D. L. Berre, P. Marquis, S. Roussel, Planning personalised museum visits., in: Proceedings of the 23rd ICAPS, 2013, pp. 380–388.

[18] G. Chen, S. Wu, J. Zhou, A. K. Tung, Automatic itinerary planning for traveling services, Knowledge and Data Engineering, IEEE Transactions on 26 (3) (2014) 514–527.

[19] I. R. Brilhante, J. A. F. de Macêdo, F. M. Nardini, R. Perego, C. Renso, Tripbuilder: A tool for recommending sightseeing tours, in: Advances in Information Retrieval - 36th European Conference on IR Research, ECIR 2014, Amsterdam, The Netherlands, April 13-16, 2014. Proceedings, 2014, pp. 771–774.

[20] I. R. Brilhante, J. A. F. de Macêdo, F. M. Nardini, R. Perego, C. Renso, On planning sightseeing tours with tripbuilder, Inf. Process. Manage. 51 (2) (2015) 1–15.

[21] A. Majid, L. Chen, H. T. Mirza, I. Hussain, G. Chen, A system for mining interesting tourist locations and travel sequences from public geo-tagged photos, Data & Knowledge Engineering.

[22] S. Jiménez, F. Fernández, D. Borrajo, Integrating planning, execution and learning to improve plan execution, Computational Intelligence Journal 29 (1) (2013) 1–36.
URL http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8640.2012.00447.x/abstract