Route Recommendations to Business Travelers Exploiting Crowd-Sourced Data

Thomas Collerton, Andrea Marrella, Massimo Mecella, and Tiziana Catarci

Sapienza Università di Roma, Italy
{collerton,marrella,mecella,catarci}@diag.uniroma1.it

Abstract. Business travellers are those people who attend work-related meetings and in their few hours of spare time would like to see the best that the host city can offer in terms of cultural activities and sightseeings. In this work we present a complex architecture, consisting of mobile applications and back-end server components, which supports business travelers in recommending possible routes matching their preferences within their timing constraints. The three main contributions are (i) a set of machine learning algorithms that can be used to detect a queuing state of a user with a high degree of accuracy, (ii) how to determine user's positioning, and (iii) how to practically realize a planner providing a reasonably good enough route plan within a handful of seconds. Preliminary tests demonstrate that the single components of the proposed architecture are feasible and provide good results.

1 Introduction

In recent years, a category of tourism has gained popularity, known as "business tourism", i.e., people who attend work-related meetings away from their hometown and in their few hours of spare time would like to see the best that the host city can offer in terms of cultural activities and sightseeings, such as monuments and museums [16]. This kind of tourism has been also fueled by the presence of emerging forms of social and technological developments that rely on sensors, big data and new ways of connectivity and exchange of information (e.g., IoT, RFID, social networks, etc.), which allow to supply business travelers with better mobility, location-based interactive services and, consequently, faster decision support [7].

The data and information intensity of business tourism has led many researchers to investigate intelligent ways to exploit ICT technologies for unlocking the power of data in order to provide more enjoyable and customized tourism experiences [24,6,25]. As a matter of fact, today's personal devices such as smartphones and tablets are almost ubiquitous, and provide arrays of sensors and antennas which can be exploited to determine various information about the user, such as her/his position and whether s/he is moving or standing still, like in a queue.

So far, ICT research in business tourism mainly provides ad-hoc case studies of existing initiatives [23,21,18] and is mostly concerned with the development of location-based technologies for tracking users' position within the cultural area (e.g., iBeacon sensors) and of mobile applications enabling travelers to share their personal experiences on social networks to help other travelers in their decision making process [7].

In this paper, we aim at presenting a general-purpose software framework called NEPTIS PLANNER, which is part of the much broader NEPTIS project [14] that focuses on developing ICT-based solutions for augmented fruition and intelligent exploration of cultural heritage. NEPTIS PLANNER offers business travelers with personalized and automatically generated routes to efficiently visit a cultural area. It is able to manage the interconnection, synchronization and concerted use of different IoT technologies for extracting crowd-sourced data from fellow tourists (e.g., to identify the waiting time to visit a specific attraction). NEPTIS PLANNER leverages on automated planning techniques [5] to reason over such data and on the traveler's preferences (e.g., the maximum available time to visit a cultural area) in order to generate on-the-fly a personalized route within the cultural area that respects the traveler's needs.

The rest of the paper is organized as follows. In Section 2 we provide an overview of the approach and architecture of the proposed system, whereas Section 3 and Section 4 present the main contributions of this work, i.e., how it is possible to detect queuing and visiting times with minimal user intervention, and how to solve a planning problem allowing the computation of a recommended route for a small and medium sized museum in a reasonable time frame, which is vital given the mobile context. Section 5 concludes the paper by discussing possible future extension of the work.

2 System overview

NEPTIS PLANNER realizes an architecture whose purpose is to provide a business traveler with a route to a set of cultural areas (e.g., museums, churches, monuments, etc.) in her/his surroundings given her/his current position and other inputs, such as a time constraint, that is how much time s/he is willing to spend for a cultural tour of the area s/he is currently in, or a list of attractions s/he would like to visit.

The system is thought for being used by two main actors:

- the *curator*, which is the user responsible for the insertion, update and deletion of information about cultural areas. For example, in the case of "open air" monuments, relevant information is their names and coordinate locations, while for "closed" museums it is relevant to know their topology (i.e., how rooms are connected between each others) and their available attractions;
- the *tourist*, which is the user/traveler who can request the compilation of a route plan within the cultural area through an application installed on her/his smartphone.

The client side of the system is provided by the tourist's mobile application (currently realized in Android), while the curator interacts with the system through a Web application. The back-end is a Node.JS server, responsible for handling requests from the clients via two sets of APIs, one for each actor. Alongside the server there is the planner component responsible for the creation of route plans, preceded by a pre-processing module which translates raw data into input files for the planner. The persistence layer has a relational database holding data inserted by the curator, as well as users' and curators' login credentials, and a data warehouse containing time intervals data coming from the tourists' smartphones. Figure 1 shows the basic workflow of the system:

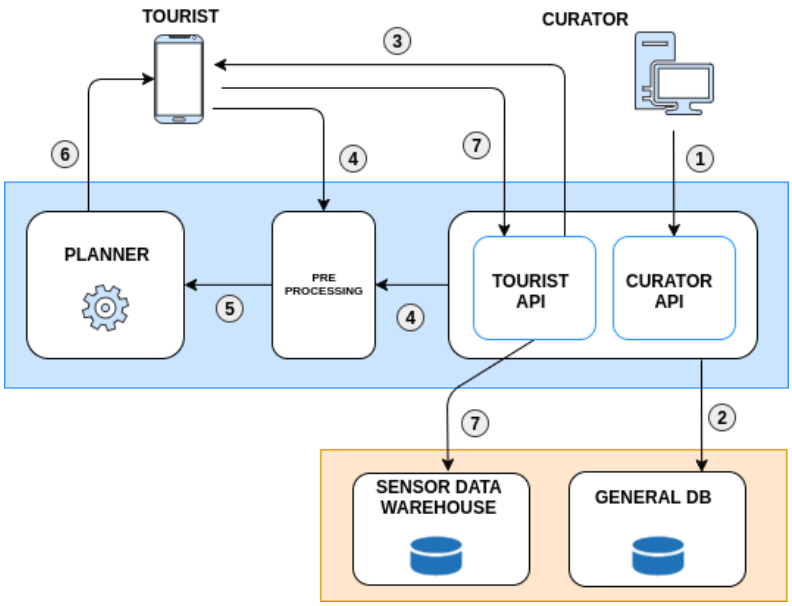

Fig. 1. NEPTIS PLANNER workflow diagram

1. The curator inserts or updates information about the cultural area s/he is responsible for. For example, this includes the list of rooms and the attractions within their premises in the case of a closed museum, or the list of monuments and their coordinates in the case of an open air museum;
2. Data are stored into a relational database;
3. The tourist mobile application exploits these data in order to help the tourist in the compilation of the request for a personalized route plan;
4. When the tourist finally completes the above step, the system feeds the needed data (both from the data warehouse and the user input) to the pre-processing module, which encodes the request for the subsequent stage;
5. The resulting files are then fed to a planning system (i.e., the planner) which will compute a cultural route plan that satisfies the constraints inserted by the tourist;
6. The output of the planning system is then sent back to the tourist, who will then start her/his own tour;
7. While the tourist is following her/his personal route plan, her/his smartphone sends updates about the time intervals needed to clear a queue or to visit an attraction; these (crowd-sourced) information is then sent to the data warehouse. Such information will be exploited for the generation of future route plans.

3 Crowd-sourcing

In order for the system to work, data about visiting times and queues are needed. In a previous iteration of the prototype, the tourist had to insert such pieces of information

manually. In general, this is a cumbersome task that can be easily forgotten and can distract the tourist from the enjoyment of the attractions s/he is visiting. In addition, the values inserted in the system can only be rough estimates rather than exact time intervals. We therefore set out to find a way to automatize the collection of data, possibly reducing user manual inputs to zero.

In the following, we will split the automatized crowd-sourcing data collection aspect in two parts: *(i)* queuing recognition; *(ii)* indoor and outdoor localization.

Queue recognition. Queuing can be an important time factor, especially in big museums where only the ticket purchase queue can last several minutes, if not hours. Although not common, some attractions can also have a lengthy waiting time before they can be fully experienced by the tourist, such as the Mona Lisa painting in the Louvre in Paris or the Sistine Chapel in the Vatican Museums in Rome. We define a user to be in a "queuing state" when s/he is either standing still or having brief bursts of movement lasting a few seconds. We take into account actions such as taking the smartphone out of the pocket or purse, taking pictures of the surroundings and similar small movements as part of the queuing state. For testing purposes, the GPS sensor included in the smartphone has been used as control and tests have been run in open fields, away from potential sources of disruption for the satellite signal. Given a certain time window Δt and a chosen threshold value of walking speed v_t, we say that a user has walked during Δt if has moved more than $v_t \cdot \Delta t$ meters.

The research literature on movement recognition is gaining a lot of momentum in the past few years, mainly due to the ever growing presence of devices such as smartphones and tablets, which pack several hardware sensors such as accelerometers, gyroscopes, magnetic compasses, etc. However, despite their popularity, the Android platform does not provide a good library for such a task; the only interface that comes close to our target is the Android Activity Recognition API[1], which is supposedly capable of guessing whether the user is walking, running or is in a car, with the help from sensors and usage data collected by Google. However, for now, tests returned lackluster results.

Therefore we chose to implement and test the approach given by Ravi in [15]: given the signals of all the three axis of the accelerometer, we sample them with a sampling frequency of 50 Hz (or 1 sample every 20 milliseconds), and store the samples in a sliding window of size 256, which gives us a time interval of 5.12 seconds. For each of these windows, we compute their average, their standard deviation and their energy, defined as $\frac{1}{|w|}\sum |f|^2$, where $|w|$ is the window size and f is a frequency value of the Fourier-transformed signal, obtained via the Fast Fourier Transform algorithm implemented in the Apache Commons Math Java library[2]. The extracted data are then used by a machine learning classifier included in the Weka data mining tool[3]. Table 1 shows the chosen algorithms and their respective degrees of accuracy with regards to the GPS signal control.

We tested some of the algorithms described in [15], both on their own and combined with others. Tests have been executed on a Google Nexus 5X and a Huwaei Honor 7.

[1] cf. `developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi`

[2] cf. `http://commons.apache.org/proper/commons-math/`

[3] cf. `http://www.cs.waikato.ac.nz/ml/weka/`

Generally speaking, tests proved that this approach is effective and yields very good levels of accuracy for all used algorithms. However classifiers such as k-NN, decision tree and decision tables on their own had issues at detecting actions such taking the smartphone out of the pocket while standing, misclassifying it as a walking movement, which brings their accuracy rate between 75 and 81%. An odd combination of them, such as the one shown in the table, alongside a majority voting decision mechanism, manages to improve the detection rate up to 90%, while the neural network, implemented by the `MultiLayerPerceptron` class in Weka, achieved the best accuracy levels. In any case, in order to minimize the rate of false positives and negatives, we do not take into account the single sample window, but rather we pick a dynamic queue of five consecutive windows and determine the final outcome by choosing the classification decided by the majority.

ALGORITHM	ACCURACY (%)
Neural Network	96
k-NN (15 neighbours)	80
k-NN (20 neighbours)	78
Decision tree (C4.5)	81
Decision tables	77
SVM	75
k-NN (15) Decision tree (C4.5) Decision tables	90

Table 1. Used classifiers and their accuracy degree

Localization. Queuing is not the only time consuming activity to take into account, but also the time needed to move between the available monuments as well as to visit them completely, or their single attractions. We describe two different scenarios, which involve outdoor and indoor localization respectively.

Let's assume a user is visiting an outdoor area, such as Piazza di Spagna in Rome. The area contains attractions such as the Barcaccia fountain, the staircase and the facade of Trinità dei Monti. We say that the user has begun her/his visit when s/he gets near the monument and that s/he is over when s/he moves away from it. To achieve this target, we implemented a geofencing algorithm which uses the Android localization service via the GPS sensor. Each monument is associated with a latitude-longitude coordinate and a radius value, therefore the user is considered to be in the visiting state when her/his distance from given coordinates is lower than the radius and vice versa.

Now let's assume that the user is in a closed environment such as a museum, where the satellite signal can be barely reached, if at all, since it requires a line-of-sight con-

ROOM	ACCURACY (%)
A1	96,67
A2	94,55
A3	96,96
A4	96,00
A5	97,14
A6	96,29
A7	97,14

Table 2. The rooms and their detected accuracy

nection. The analyzed literature on the subject of indoor tracking and localization focuses on three methods:

- image recognition, where the position of the device can be guessed from its camera. A detailed study can be found in [1] and a basic implementation in the context of cultural heritage is described in [17], however the proposed method requires the user to have more or less the smartphone in her/his hands and potentially on the upright position, which can get rather uncomfortable;
- analysis of sensors' data such as accelerometers and magnetic compass [10]. We tested very briefly this method, however the sensors in the tested smartphones were not precise enough and users' actions can be random enough to generate some noise to make the readings even less precise;
- RSS (Radio Signal Strength) of multiple WiFi signals from various access points, which allows a classification based upon the triangulation of these signals [8]. This requires a training set of all the interested areas.

We approached the latter case, because it requires the least amount of input from the user and, provided that the location has some WiFi access points, can provide interesting results. We tested this approach with FIND[4], an open-source software which implements a server that stores RSS fingerprints and classifies incoming beacons with a Naive Bayes or SVM algorithm. We used the ground floor of our department as a test bed, treating the area in front of the seven available rooms as a wall with exposed paintings, both in a crowded and almost empty corridor. The device used for this test is a Google Nexus 5X. Table 2 shows the detection accuracy for each of the rooms. All the tested rooms have been detected with an accuracy exceeding 95% with the exception of room A2, which is slightly lower due to it being very close to room A1, which was our target threshold, and therefore we can consider FIND a suitable tool for our target.

4 On the synthesis of route plans

Given the time intervals data received from the smartphones of the users visiting a cultural area C, the target is to provide a business traveler that wants to visit C with

[4] cf. `http://www.internalpositioning.com/`

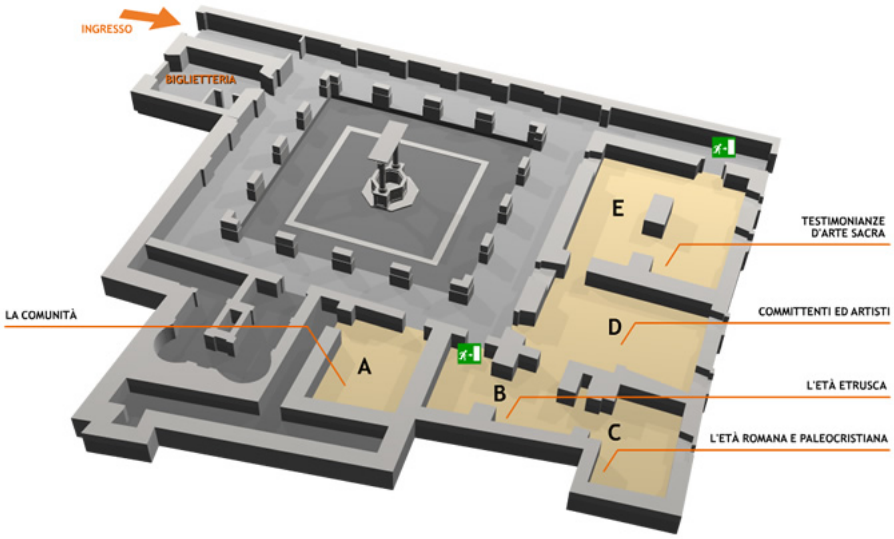

Fig. 2. A sketch of the floor plan of the local civic museum of Bracciano.

a route plan R that can be completed in the *minimum time required to maximize the traveler's needs and preferences*.

The problem can be represented through a directed graph $G(V, E)$, which holds information about the topology of C. We call such a graph a *topology graph*. Each node $v \in V$ represents a single distinguishable *zone* of C (e.g., if C is a museum, zones correspond to museum's rooms), i.e., a cluster of *attractions* belonging to v. There is a special "dummy" node labeled as "Entry/Exit" that defines the entry/exit point of C.

Edges of G are of two kinds, E_{conn} and E_{att}, with $E_{conn} \cup E_{att} = E$ and $E_{conn} \cap E_{att} = \emptyset$. Each edge $e_x \in E_{conn}$ is used to represent a *physical connection* (be it a corridor, a door, a staircase) between two different nodes $v_i, v_j \in V$, and is labeled with the estimated time t_{conn_x} required to move between them. On the other hand, any edge $e_y \in E_{att}$ that insists on the same node $v \in V$ (i.e., a *loop* edge) represents the *visit* of a single attraction located in v. Loop edges e_y are labeled with a certain *score* w_y, which is calculated taking into consideration the average rating rt_y given by the users to the attraction and the time needed to visit it (i.e. $t_{queue_y} + t_{visit_y}$).

The rating of an attraction can assume values in the {1,5,10} domain, while the times are expressed in minutes. We assume that if an attraction has a long queue or visit time, then its visit is considered more important than some other which has perhaps the same rating. Consequently, we calculate w_y such that $w_y = rt_y \cdot (\frac{1}{t_{queue_y} + t_{visit_y}} \cdot 100)$. In this way, it follows that the smaller the value of w_y, the more relevant is the visit of the attraction labeled with w_y.

As a running example, Figure 2 shows the floor plan of the local civic museum of Bracciano, a town north of Rome which has been used for some tests of this work,

and the left-hand part of Figure 3 shows its graph representation. Notice that in the running example each node has one loop edge, each representing an attraction in that zone (labelled for simplicity as $a0, a1, a2, a3, a4$).

Given the above model, the problem can be defined as *finding a path in the topology graph that minimizes the total score of visited attractions and the time for moving between different zones*. This allows — consequently — to minimize the overall time required to a business tourist for a satisfactory visit of C. This problem is known in literature as the ORIENTEERING PROBLEM, a variant of the Traveling Salesman Problem (TSP) where the objective function is to maximize or minimize the total value associated to each node of the graph.

The problem is known to be NP-hard, as well as APX-hard, and some heuristic and approximation algorithms, described in [22], [20] and [4] have been studied, even if no efficient implementations, to the best of our knowledge, are available.

In order to tackle this issue and generate quality route plans in a reasonable time, we decided to adopt techniques coming from the *automated planning* field, which is a branch of Artificial Intelligence (AI) that aims to the realization of automated systems for the synthesis of organized sequences of real-world activities [5]. To this aim, in the following sections we first show some preliminaries on automated planning necessary to understand the rest of the paper, and then we present our solution to encode the problem of synthesizing a route plan as a planning problem in AI, which can be solved by state-of-the-art planners. Finally, we discuss some experiments performed through two state-of-the-art planners that demonstrate the versatility of our approach and its feasibility in realistic settings.

Basics of automated planning. Automated planning operates on explicit representations of states and actions. The Planning Domain Definition Language (PDDL) is a de-facto standard to formulate a *planning problem* $\mathcal{P} = \langle I, GL, \mathcal{P}_\mathcal{D} \rangle$, where I is the initial state of the world, GL is the desired goal state, and $\mathcal{P}_\mathcal{D}$ is the planning domain.

A planning domain $\mathcal{P}_\mathcal{D}$ is built from a set of *propositions* describing the state of the world and a set of *actions* Ω that can be executed in the domain. An *action schema* $a \in \Omega$ is of the form $a = \langle Par_a, Pre_a, Eff_a \rangle$, where Par_a is the list of *input parameters* for a, Pre_a defines the *preconditions* under which a can be executed, and Eff_a specifies the *effects* of a on the state of the world. Both preconditions and effects are stated in terms of the *propositions* in $\mathcal{P}_\mathcal{D}$. Propositions can be represented through boolean predicates and fluents. In the remainder of the paper, we remain consistent with PDDL terminology [3]: a *predicate* is a boolean property of the world and *fluents* are used to express numeric properties, such as the actions' cost. Both the values of predicates and fluents can change as result of the execution of actions. PDDL includes also the ability of *typing* the parameters that appear in actions and *constraining* the types of arguments to predicates and fluents.

There exist several forms of planning in the AI literature. In this paper, we focus on planning techniques characterized by *fully observable*, *static* and *deterministic* domains, i.e., we rely on the classical planning assumption of a "perfect world description" [26]. Concretely, this implies that: *(i)* any planning action only provides deterministic and observable effects; *(ii)* a complete knowledge of the initial state I is available.

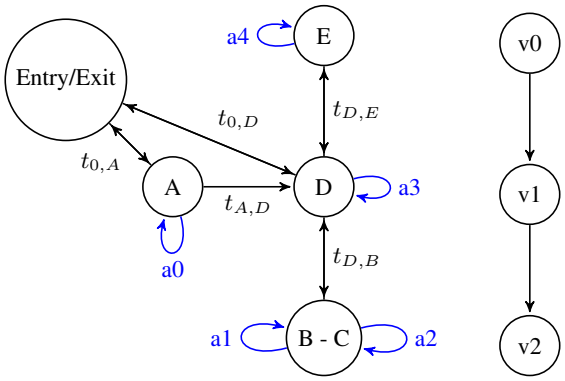

Fig. 3. The museum's floor plan graph remodeled for the planning problem.

A solution to a planning problem is a sequence of actions—*a plan*—whose execution brings from initial state I to some state that satisfy goal GL. The plan is said to be *optimal* if it minimizes the sum of action costs. Automated planning has made huge advances in the last twenty years, leading to solvers able to create plans with thousands of actions for problems described by hundreds of propositions (see, for example, [12,2,13,11]). In this work, we represent planning domains and problems making use of the STRIPS fragment of PDDL 2.1 [3], enhanced with the numeric features provided by the "level 2" of the same language. Such features are used to keep track of the costs of planning actions and to synthesize plans satisfying pre-specified metrics.

Encoding as a planning problem. In order to encode the problem of synthesizing a route plan as a planning problem in PDDL, in addition to the topology graph we made use of a second graph containing as many nodes as are the number of attractions that the traveler would like to visit (cf. the right-hand part of Figure 3). We call such a graph a *visit graph*, and its use is devoted to avoid the generation of route plans containing empty lists of attractions.

In the planning domain $\mathcal{P}_\mathcal{D}$, we provide two abstract types called `attraction` and `node`. The first captures the attractions involved in the loop edges of the topology graph. The second is used to identify the nodes of the topology graph (through the sub-type `topology_node`) and of the visit graph (through the sub-type `visit_node`). To capture the structure of the topology/visit graph and to monitor their evolution, we defined four *domain propositions* as boolean predicates in $\mathcal{P}_\mathcal{D}$[5]:

- `(cur_node ?n - node)` holds if `n` is the current node of a topology/visit graph.
- `(edge ?n1 - node ?n2 - node)` holds if there exists an edge from two different nodes `n1` to `n2` of a topology/visit graph.
- `(visited ?a - attraction)` holds if the attraction `a` is visited.

[5] Variables are distinguished by a "*?*" character at front, and the dash "–" is used to assign types to the variables.

– `(at ?a - attraction ?n - topology_node)` indicates that the attraction `a` is located in a zone represented by the node `n`.

Furthermore, we define three *numeric fluents* in $\mathcal{P}_\mathcal{D}$: *(i)* `(move-cost ?n1 - topology_node ?n2 - topology_node)` records the time required to move from `n1` to `n2`; *(ii)* `(visit-cost ?a - attraction)` reflects the score associated to the visit of the attraction `a`; *(iii)* `(total-cost)` keeps track of the overall cost of the route plan under construction.

In the planning problem $\mathcal{P}$, we first define a finite set of constants required to properly ground all the domain propositions defined in $\mathcal{P}_\mathcal{D}$. In our case, constants will correspond to the nodes and attractions instances involved in the topology/visit graph. Secondly, we define the *initial state* of $\mathcal{P}$ to capture the exact structure of the topology/visit graph. This includes the specification of the current nodes of the graphs (in I, the current nodes of the topology and of the visit graph are "Entry/Exit" and $v0$, resp.) and of all the existing edges that connect two different nodes of the graphs. Furthermore, in I the exact location of any attraction is specified, while it is assumed that no attraction has been yet visited. Thirdly, we define the goal condition GL as the conjunction of the *final nodes* of the two graphs. The final node of the topology graph is (again) "Entry/Exit", while the final node of the visit graph depends by the number of attractions that the traveler aims to visit; in our running example, it is $v2$ (cf. Figure 3).

The plan to reach GL from I is constituted by a sequence of planning actions that allow to perform movements between nodes in the two graphs and to eventually reach the final nodes. Specifically, in $\mathcal{P}_\mathcal{D}$ we provide two planning actions that allow: *(i)* to move between two nodes of the topology graph, or *(ii)* to visit an attraction (not yet visited) of the topology graph; the latter allows also to move forward in the visit graph.

```
(:action move
 :parameters(?n1 - topology_node ?n2 - topology_node)
 :precondition (and (cur_node ?n1) (edge ?n1 ?n2))
 :effect (and (not (cur_node ?n1)) (cur_node ?n2)
              (increase (total-cost) (move-cost ?n1 ?n2))))

(:action visit
 :parameters(?a1 - attraction ?s1 - topology_node
             ?v1 - visit_node ?v2 - visit_node)
 :precondition (and (cur_node ?n1) (at ?a1 ?n1) (not (visited ?a1))
                    (cur_node ?v1) (edge ?v1 ?v2))
 :effect (and (not (cur_node ?v1)) (cur_node ?v2) (visited ?a1)
              (increase (total-cost) (visit-cost ?a1))))
```

Readers should finally notice that the execution of the `move` (or `visit`) action makes total cost of the route plan increases of a value equal to `(move-cost)` (or `(visit-cost)`). Since our purpose is to minimize the total cost of the generated route plan, the planning problem also contains the following specification: `(:metric minimize (total-cost))`.

Preliminary validation. The approach has been positively assessed using automatically generated PDDL files that describe problems of growing complexity. We performed our experiments with a machine equipped with an Intel Core i7-4770S CPU

3.10GHz Quad Core and 16GB RAM. The ideal target was to compute *optimal routes*, i.e., route plans that allow a traveler to minimize the time required for a visit by maximizing its quality (in terms of the ratings of the visited attractions).

To this aim, we performed experiments by making use of of the SymBA*-2 [19] planning system (winner of the sequential optimizing track at the 2014 International Planning Competition), which performs a bidirectional A* search to find optimal plans.

However, since automated planning is known to be PSPACE-complete and optimal algorithms can take a major toll on computation time [5], we decided to run further experiments using a sub-optimal heuristic based on FF (that is, forward chaining heuristic, see `http://www.fast-downward.org/Doc/Heuristic`) running on the Fast Downward planning system [9], in order to return near-optimal solutions (i.e., solutions with less quality than optimal ones) in a shorter time.

To have a sense of the scalability with respect to the size of the problem, we generated PDDL files representing topology graphs of growing complexity. Specifically, topology graphs used for the experiments have the following properties:

- a number of nodes varying from 10 to 30;
- each node can host from 1 to 15 attractions;
- each node can be connected just to another node (like in a single link chain), or it has 50%-30% chance of being connected to two-three other nodes.

In addition, we customized the number of attractions to be visited in the visit graph and those that must be necessarily included in the route, as follows: *(i)* from 5 to 50 attractions; *(ii)* from 0 to 5 randomly chosen "must visit" attractions[6].

Figure 4 shows the computation time comparison between the two planning algorithms on four selected problems, specifically those which describe a graph with 10 nodes, 5 to 25 selected visits and a variable number of attractions that have been selected as a "must visit". For each problem, we wanted to monitor the performances of the planning systems given the number of attractions (on the x axis) and the complexity of the modeled area (that is, the number of connections for each node, one plot for each kind of connection; therefore each algorithm has three plots associated with it).

Tests show that FF manages to maintain a stable performance and stay below 3 seconds, while SymBA*-2 has a more random pattern and can take 100 times longer than FF. SymBA*-2 tends to have an exponential growth as the number of attractions and nodes increase. Even in the case of larger domains, such as the ones shown in table 3, FF can come up with a solution in less than 30 seconds, which is the target maximum wait time which we would like to achieve.

Figure 5 shows the total score of the selected attractions for the same problems; while SymBA*-2, which is based upon an optimal A* algorithm, always returns the best result, FF is never far off from it, managing to even get the exact same value. Therefore we can say that, between FF and SymBA*-2, the earlier is the more appealing solution.

The reader should notice that the power of a planning-based approach is also linked to its versatility. Starting from a PDDL encoding, the approach allows one to plug in new planning algorithms/systems at basically no cost. In the future we can improve the performance even further if a better planner will be released.

[6] Must visit attractions are specific attractions that a user explicitly asks to visit and that must appear in the generated route plan.

ROOMS	ATTRACTIONS	VISIT	TIME (SECONDS)
15	75	25	1,08
20	140	25	2,27
20	220	15	3,40
30	150	25	2,33
30	330	25	7,28

Table 3. Execution times for some of the performed tests

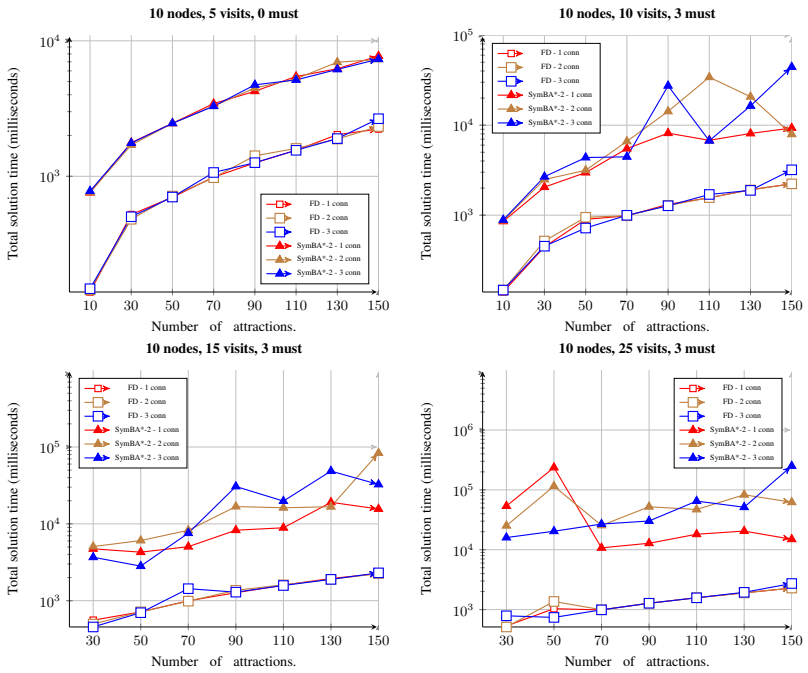

Fig. 4. Comparison of computation times between FF and SymBA*-2

5 Concluding remarks

In this work we have shown that a set of machine learning algorithms can be used to detect a queuing state of a user with a high degree of accuracy, a software such as FIND can determine the user's positioning within a room and a planning system can provide a reasonably good enough route plan within a handful of seconds, all of this in order to support tourists with time constraints (such as business travellers). Thus far,

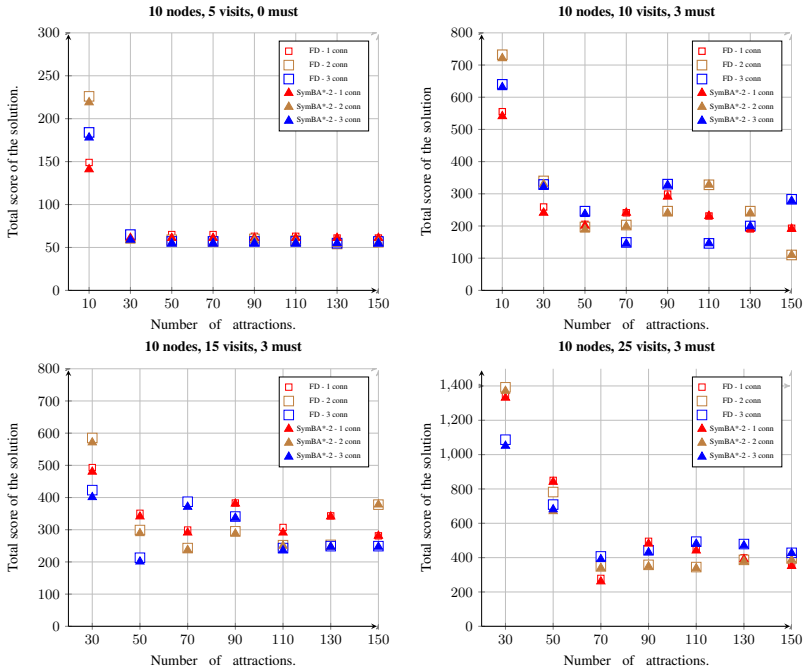

Fig. 5. Comparison of the quality of the solutions obtained by FF and SymBA*-2

preliminary tests have demonstrated that the single components of our architecture are feasible and provide good results. Validation tests (with a large number of users) on the integrated system have not yet been performed, and will be carried out during the 2nd half of 2017, in the context of the NEPTIS project. However we are confident that the completed system will provide good enough results to be usable within its defined use cases. Given also the ever growing computational power of mobile devices and accuracy of their sensors, new detection and localization techniques will be able to improve the accuracy levels that we tested, thus making the framework even more precise.

Acknowledgments. This work has been supported by *NEPTIS – PON03_PE_0214*.

References

1. Aicardi, I., Dabove, P., Lingua, A.M., Piras, M.: Sensors integration for smartphone navigation: performances and future challenges. In: ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. vol. XL-3, pp. 9–16 (2014)
2. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the Disruptive Effectiveness of Automated Planning for LTL$_f$-Based Trace Alignment. In: Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17). pp. 3555–3561 (2017)
3. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. J. Artif. Intell. Res. (JAIR) 20(1), 61–124 (2003)

4. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G.: A survey on algorithmic approaches for solving tourist trip design problems. Journal of Heuristics 20(3) (2014)
5. Geffner, H., Bonet, B.: A concise introduction to models and methods for automated planning. Synthesis Lectures on Artificial Intelligence and Machine Learning 8(1), 1–141 (2013)
6. Gretzel, U.: Intelligent systems in tourism: A social science perspective. Annals of Tourism Research 38(3), 757–779 (2011)
7. Gretzel, U., Sigala, M., Xiang, Z., Koo, C.: Smart tourism: foundations and developments. Electronic Markets 25(3), 179–188 (2015)
8. Hatami, A., Pahlavan, K.: A comparative performance evaluation of RSS-based positioning algorithms used in WLAN networks. In: IEEE Wireless Communications and Networking Conference. vol. 4 (March 2005)
9. Helmert, M.: The Fast Downward Planning System. J. Artif. Intell. Res.(JAIR) 26 (2006)
10. Jin, Y., Toh, H.S., Soh, W.S., Wong, W.C.: A robust dead-reckoning pedestrian tracking system with low cost sensors. In: 2011 IEEE International Conference on Pervasive Computing and Communications (PerCom). pp. 222–230 (March 2011)
11. de Leoni, M., Marrella, A.: Aligning Real Process Executions and Prescriptive Process through Automated Planning. Expert System with Applications 82, 162 – 183 (2017)
12. Marrella, A., Lespérance, Y.: Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms. In: 14th Int. Conf. on Business Process Modeling, Development, and Support (BPMDS). pp. 277–291 (2013)
13. Marrella, A., Mecella, M., Sardiña, S.: Intelligent Process Adaptation in the SmartPM System. ACM TIST 8(2), 25:1–25:43 (2017)
14. di Palermo, U.: Neptis. Soluzioni ICT per la fruizione e l'esplorazione "aumentata" di beni culturali (2015)
15. Ravi, N., Dandekar, N., Mysore, P., Littman, M.L.: Activity recognition from accelerometer data. In: 17th Conference on Innovative Applications of Artificial Intelligence - Volume 3. pp. 1541–1546. IAAI'05, AAAI Press (2005)
16. Robinson, P.: Tourism: The key concepts. Routledge (2012)
17. Rubino, I., Barberis, C., Xhembulla, J., Malnati, G.: Integrating a Location-Based Mobile Game in the Museum Visit: Evaluating Visitors Behaviour and Learning. J. Comput. Cult. Herit. 8(3), 15:1–15:18 (May 2015)
18. Sigala, M., Chalkiti, K.: Investigating the exploitation of web 2.0 for knowledge management in the Greek tourism industry: An utilisation–importance analysis. Computers in Human Behavior 30, 800–812 (2014)
19. Torralba, A., Alcazar, V., Borrajo, D., Kissmann, P., Edelkamp, S.: Symba: A symbolic bidirectional planner. In: International Planning Competition. pp. 105–108 (2014)
20. Toth, P., Vigo, D.: Vehicle routing: problems, methods, and applications. SIAM (2014)
21. Tu, Q., Liu, A.: Framework of smart tourism research and related progress in China. In: International Conference on Management and Engineering (CME 2014). pp. 140–146 (2014)
22. Vansteenwegen, P., Souffriau, W., Oudheusden, D.V.: The orienteering problem: A survey. European Journal of Operational Research 209(1), 1 – 10 (2011)
23. Venturini, A., Ricci, F.: Applying Trip@dvice Recommendation Technology to www.visiteurope.com. Frontiers in Artificial Intelligence and Applications 141, 607 (2006)
24. Werthner, H.: Intelligent systems in travel and tourism. 18th International Joint Conference on Artificial Intelligence (IJCAI 2003) (2002)
25. Werthner, H., Alzua-Sorzabal, A., Cantoni, L., Dickinger, A., Gretzel, U., Jannach, D., Neidhardt, J., Pröll, B., Ricci, F., Scaglione, M., et al.: Future research issues in it and tourism. Information Technology & Tourism 15(1), 1–15 (2015)
26. Wilkins, D.E.: Practical planning: extending the classical AI planning paradigm. Morgan Kaufmann (1988)