RC25547 (IRE1505-014) May 25, 2015
Computer Science

IBM Research Report

Docit: An Integrated System for Risk-Averse Multi-Modal Journey Advising

Adi Botea, Michele Berlingerio, Stefano Braghin, Eric Bouillet, Francesco Calabrese, Bei Chen, Yiannis Gkoufas, Rahul Nair
IBM Research
Smarter Cities Technology Centre
Mulhuddart
Dublin 15, Ireland

Tim Nonner, Marco Laummans
IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research Division
Almaden – Austin – Beijing – Cambridge – Dublin - Haifa – India – Melbourne - T.J. Watson – Tokyo - Zurich

Docit: An Integrated System for Risk-Averse Multi-Modal Journey Advising

Adi Botea, Michele Berlingerio, Stefano Braghin
Eric Bouillet, Francesco Calabrese, Bei Chen
Yiannis Gkoufas, Rahul Nair
IBM Research, Dublin

Tim Nonner, Marco Laummans
IBM Research, Zurich

Abstract

Current systems for multi-modal journey planning assume a deterministic environment. However, in reality, transportation networks feature many types of uncertainty, such as variations in the arrival times of public transport vehicles. Slight errors in the deterministic assumptions can result in lost connections, with a corresponding delay at the arrival.

We present Docit, the first multi-modal journey advising system that reasons about uncertainty in the network knowledge, creating journey plans optimized on the likelihood of arriving on time. We describe its main functions, created both for travellers and network operators. We discuss our solutions to integration challenges, including the integration, as part of the same system, of two different uncertainty-aware planning engines. Our system has been integrated with two commercial products, to gain access to dynamically updated network data, and to provide network operators with network awareness information computed by our system.

1 Introduction

In many cities, the increasing traffic of private and commercial vehicles is straining the transportation infrastructure, and traffic congestion causes significant losses to the economy. Public transport has a significant potential to reduce traffic and congestion, but only if a substantial modal shift can be achieved. Reasons for the still relatively low adoption of public transport include its perceived unreliability and inconvenience for spontaneous travel.

Unreliability is due to the inherent uncertainty in a public transport system, where vehicles might deviate from their planned schedule due to all kinds of disturbances. Small initial disturbances are amplified by the connections between services, which are crucial for a network's connectivity but can lead to significant passenger delay due to missed connections. Inconvenience is related to passengers needing a considerable lead time to pre-plan their journey as the level of service and connectivity is not uniform over time. In this context, suitable journey advice, for planning the trip as well as guiding the passenger during the trip, is an important enabling factor for better public transportation service.

Journey planning is the process of advising travelers on how to best use a given transportation system to their journey requests. It is standard practice for transport operators to offer journey planning applications either on their websites or via mobile apps. For a given journey request, these applications typically return one or several itineraries as linear sequences of activities from start to destination, based on efficient shortest path computation engines working on a time-expanded graph model of the service network.

Current journey planning technology assumes a deterministic environment and hence uses a deterministic model. However, due to changing traffic conditions or other disturbances, public transport vehicles are not always on schedule. Any planned itinerary may become suboptimal or even infeasible, for example due to missed connections. To compensate this shortcoming of pre-planned itineraries, modern transport operators have started to offer push-services to travelers, informing them of missed connections together with a new, updated itinerary based on the current situation. Although this offers some degree of adaptability, it is obviously only a reactive approach, based on deterministic planning applied on the basis of the current situation.

Stochastic planning can take contingencies into account during plan computation. A typical public transport network offers multiple alternative services for a traveler to choose at a given location which provide the necessary flexibility to react to disturbances and are the natural basis for a policy-based, uncertainty-aware journey planning (Botea et al. 2013; Nonner 2012; Nonner and Laumanns 2014). A policy is a time-, state-, or history-dependent routing advise at each location. Different classes of policies can be distinguished based on the considered information or the set of actions considered, i.e., whether they specify a single service or a set of services in each state. Algorithms to find optimal policies include AO* (Nilsson 1968) based search methods, combinatorial approaches or dynamic programming, which work well on instances of realistic size, even though the problem is NP-hard for most policy classes. Resolving the algorithmic challenges for finding policy-based journey plans is a relatively new and active research area.

Building and deploying a system for uncertainty-aware journey planning involves a whole set of new questions regarding data collection and preparation, building appropriate stochastic models, formalization of the notion of journey plans as policies, integration of sophisticated methods for planning under uncertainty, real-time tracking of the state of the system and the travelers, user interaction and interface design.

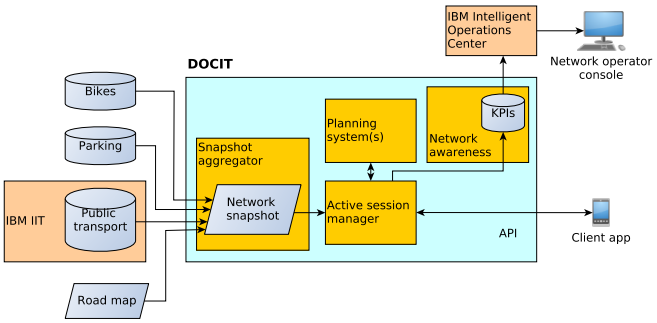

Figure 1: Overview of the system architecture.

In this paper we discuss these design questions and outline the main considerations and choices we have made to implement the *first* uncertainty-aware multi-modal journey planning system in practice. We have faced multiple integration aspects, such as the integration of our system with commercial products, such as the IBM Intelligent Transportation (IIT)[1] and the IBM Intelligent Operations Center (IOC),[2] the aggregation of heterogeneous transportation data into a unitary knowledge base, and the ability to plug in multiple uncertainty-aware journey planning systems such as DIJA (Botea et al. 2013) and the Frequency Planner (Nonner 2012; Nonner and Laumanns 2014), facilitated by the design of a policy format generic enough to capture the different semantics and syntax of different types of policies.

2 System Overview

Figure 1 illustrates the architecture of our system, which we call Docit. This section overviews the main components, their main functions, and their interactions with each other, with external apps and with commercial products.

In Figure 1, the Snapshot Aggregator creates a knowledge base with all the information available about a multi-modal transportation network. The resulting knowledge base is called the *network snapshot*, or simply the snapshot. As discussed later, the snapshot is key input data to important system functions, such as journey planning and journey monitoring. Challenges associated with the data aggregation and the way we address them are discussed in Section 3.

The main functions of the Active Session Manager (ASM) include keeping a record of all travellers currently using the system (the *active sessions*), communicating with travellers, communicating with the planning engine, and providing both current and historical data for the computation of key performance indicators (KPIs). For example, one KPI based on historical data can identify hotspots on the map where travellers miss connections relatively frequently. Hotspots can be presented to the network operators. Experts can take this into account when performing network optimisation functions, to better synchronise the two transport lines involved in the missed-connection pattern.

When dynamic network updates are available, the ASM regularly receives a new network snapshot from the Snapshot Aggregator. The IBM IIT product can provide real-time updates on the estimated times of arrival for public transport vehicles, which is a main reason why we have worked on integrating Docit with IBM IIT. In addition, dynamic updates on bike station and car parking data, available in a separate database, are used to estimate waiting times until a bike, a bike parking spot or a car parking spot become available.

Travellers use client apps to interact with the system. Client apps communicate with the ASM through an API. The main functions include submitting journey plan requests, receiving plans, submitting updates on the progress of a trip, submitting requests on the validity status of a plan given the most recent network information available, and receiving notifications on the plan validity status.

Consider that a traveller submits a new journey plan request through a client app, such as a mobile app or a web-based app. The ASM starts an active session for that user. It forwards the request, together with a pointer to the most recent network snapshot available, to the planning engine, and it receives one or several journey plans in response.

After accepting one journey plan, the user can opt for a journey monitoring function, which allows to receive notifications about plan invalidations caused by dynamic events in the transportation network, and to perform replanning. See a detailed example in Section 6.

Brute-force replanning (i.e., computing a new plan for every active user every time a new snapshot is available) is undesirable for two reasons. First off, suggesting a change of plans too frequently, even when the old plan remains perfectly acceptable, can reduce the level of user satisfaction. Secondly, replanning for *all* users every time might cause a scalability bottleneck.

In addressing these limitations, our approach to replanning works as follows. Every time the ASM receives a new snapshot, it checks whether the current active plans (one plan for each active user) remain valid, given the refreshed knowledge. The check involves simulating ahead the rest of the plan, with the new network snapshot, obtained from the Snapshot Aggregator, and the current position of the traveller along the plan, obtained regularly from the mobile app.

The simulation procedure returns a yes/no answer about whether the destination will still be reached in time, given the refreshed data. In a policy (plan) with one or more pathways from a current state to the destination, each individual pathway is simulated ahead. At each connection point, the simulation will take the next available trip on the route stated in the pathway. If no such trip is available (e.g., a delay caused missing the previous trip, and no other trips are scheduled), then the simulation of the pathway fails. Otherwise, if the simulated arrival time at the destination exceeds the initially provided arrival time by more than a user-specified threshold, the simulation fails as well. In all other cases, the simulation of the pathway at hand succeeds.

In a multi-pathway policy, it is acceptable that one or more pathways fail, as long as the traveller is guaranteed

[1] http://www.ibm.com/software/products/en/intelligent-transportation/

[2] http://www.ibm.com/software/products/en/intelligent-operations-center/

to reach the destination in time along one of the remaining pathways. If no such a guarantee is provided, the simulation of the entire policy fails, and the policy is considered invalid.

Simulating the steps of a plan is much cheaper than computing a plan. The simulation time is linear in the plan size, whereas searching for a plan often is exponential in the plan size. Our system replans only for the invalidated plans, with a corresponding reduction in the computational effort.

Docit is focused on risk-averse journey planning, considering the uncertainty present in the network data, and computing plans with a better chance of arriving in time, in case of mishaps such as missed connections. We have integrated two risk-averse journey planning engines, DIJA (Botea et al. 2013) and the Frequency Planner (Nonner 2012), whose sources were provided by their respective authors. A detailed discussion of the planning algorithms is beyond the focus of this short paper, where we discuss integration aspects of the architecture.

As detailed in Section 4, the two planning engines work with significantly different types of policies. From a system integration perspective, the challenge is to design a policy format that is sufficiently general to cover the specific types of policies implemented in various planning systems, such as the two planners integrated into our architecture. A generic policy format offers the freedom to plug in any planning engine that complies with the format. Policy integration aspects are discussed in Section 4.

The ASM provides data to the Network Awareness module, such as the trajectories of active travellers, and statistics about completed sessions. The latter includes, for each session, the journey request, the actual trajectory followed, the arrival time, as well as plan invalidations and replanning rounds experienced in the session at hand. The data is visualized onto the network operator console. We have implemented two ways of visualizing such data: a standalone application, called the Dashboard, and visualisation functions in the user interface of the IBM IOC product. See more details in Section 5.

For the use of travellers, we have developped both a mobile app and a web-based app. Both apps allow the user to enter a journey plan request, after which the request is submitted to the Docit system. Policies (journey plans) are visualized both on a map and separately, as a graph of locations and journey legs, with details about the timing, the transport modes to follow and the routes to follow on each public transport link. The mobile app implements additional functions, needed for journey monitoring, such as notifying the server on the actual branch followed by the traveller on a multiple-branch policy, receiving updates of the valid/invalid status of the plan, requesting replanning and receiving new plans. A detailed discussion of the client apps is beyond the focus of this short paper.

3 Aggregating Network Data

The network snapshot combines data about the public transport, the road map, car parking lots and bike stations in a shared-bike network.

When aggregating multi-modal transportation data, the diversity of the data refers not only to the different types of uni-modal subsets, but also to the fact that the same kind of data can originate from different sources.

Take, for example, *predictive data*, which is data that covers a time window in the future, such as the remaining part of a day. In multi-modal transportation, examples include: 1) The estimated times of arrival (ETAs) and departure of public-transport scheduled vehicles (e.g., buses), at various times of the day. 2) The availability of free spots in a car parking lot, at different times of the day. When the predicted availability is zero, also predict the waiting time until a spot will become available. 3) The availability of bikes and free parking spots, at a bike station, at various times of the day. When the predicted availability is zero, include a prediction of the waiting time until the desired resource (bike or parking spot) becomes available. 4) The predicted travel speed along the segments of a road map.

Predictive data can be obtained from various sources, each one having its own benefits and drawbacks. For example, published schedules (timetables) are one source of public transport ETAs. These have the advantage that they are increasingly available to download, being provided by transportation agencies and city authorities. They cover large time windows, being able to offer long-term predictive data. On the other hand, published schedules are static. They lack stochastic information to model the uncertainty in the ETAs. They also lack accurracy, in those cases when an actual bus trip does not closely respect the published timetable.

Stochastic noise information can be compiled from the second data source we discuss, namely historical data about actual arrival times. This is potentially more accurate than static timetables, but it is less frequently available as well. The availability depends on the existence of a GPS-reporting infrastructure, as well as the willingness of transport agencies to make the data with the actual journeys available.

Finally, ETAs can also be computed in real time, from GSP data collected from public transport vehicles moving around the city. The advantage is an increased accurracy of the results. The disadvantage is that such a functionality is still relatively uncommon. Furthermore, by their nature, real-time predictions are available only for a short time window into the future. IBM IIT can provide real-time updates on the estimated times of arrival and departure, as well as historical statistics of these. This functionality works only for vehicles that have already started their journey, which in turn limits the prediction time window.

Given the strengths and the drawbacks of each data source, combining multiple sources, when they are available, is a natural idea. The resulting snapshot is agnostic to where the data comes from. The structure is the same, and there is no separation line between say, ETAs computed in real time and static, timetable ETAs. When only one subset of the sources are available (e.g., just timetable data, without real-time predictions), the system is still able to build a complete snapshot.

For car parking and bike data we have implemented the prediction algorithm reported by Chen et al. (2013). The planning engines we use in our system make use of the waiting time until a bike or parking spot becomes available, not of the predicted number of bikes or parking spots. Thus, the

ability of providing waiting time predictions was a key requirement to be able to integrate the prediction method and the planning engine as part of the same system.

In the prediction method our assumption is that the inter-arrival times follow an exponential distribution with time-varying intensity $\lambda(t)$ at time t, $t \in \mathbb{Z}^+$. Equivalently, we assume that the parking/bike arrivals follow an inhomogeneous Poisson process. This is because empirical evidence shows that the arrival intensity is a function of time, i.e., it is particularly high during busy periods, such as morning or evening rush hours; while it is relatively low in off-peak hours, such as late at night or early morning. We use the following procedure to estimate the $\lambda(t)$ for a given time t, illustrated here for the case of bike prediction.

Days in the historical data are split into two categories, one for working days and one for weekends and public holidays. If desired, a finer separation, including weather data, for instance, can be performed. The following steps are applied separately for each category.

- For each day in the historical data, calculate the duration of all inter-arrival periods and denote them by d_1,d_2,..., d_k. Also count the bikes arrived minus the bikes departed in every period, and denote these by n_1, n_2, ..., n_k.
- Compute the estimated arrival intensity as the number of increments per unit of time, $\hat{\lambda}(t) = n_{i(t)}/d_{i(t)}$, where $i(t)$ is the index of the inter-arrival period into which t falls. If no such period exists (i.e., t lies before the first or after the last arrival on that particular day), then set $\hat{\lambda}(t) = 0$.
- Compute the average of all estimates $\hat{\lambda}(t)$ obtained for each day in the training set.

Consequently, the estimated distribution of the waiting time is exponential with mean equal to the average $\hat{\lambda}(t)$.

Data format

We have adopted a csv-like data format for the network snapshot. Our system queries the data sources available, such as relational databases or files on the disk, and builds a unitary, source-agnostic snapshot as a series of csv files. The snapshot is used in the computation of journey plans, and in the validity checks of existing plans.

Public transport data is based on GTFS,[3] a csv-based format including data such as stops, routes, and trips. Stops are characterized by an id, a name, and lat/lon coordinates. Each route is served by multiple trips, where trips are actual vehicle journeys along that route. Each trip is an ordered sequence of stops, with arrival and departure times associated with each stop. Standard GTFS supports deterministic arrival and departure times. We have extended the format so that arrival and departure times can optionally include a stochastic noise, modelling uncertainty in the arrival or departure times. The noise is an extra column, of type string, in the csv file of trip timing data. For instance, a value such as "N(0,6400)" represents a Normal distribution with a mean value $\mu = 0$ and the variance $\sigma^2 = 6400$.

[3] https://developers.google.com/transit/gtfs/

As GTFS is restricted to public transport, we have extended the format to model road maps, bike station data, and car parking data.

Car parking data and bike station data have a similar structure with each other. A list of car parking lots stores, for each record, as mandatory fields, a name, an id, lat/lon coordinates, and one or several links between the parking lot and nearby nodes from the road map. The format allows to use optional fields, such as the total capacity. We represent predicted waiting times for a parking spot as records in a csv file, each record containing the id of a car parking lot, a discrete, determistic time of arrival at that parking lot, and a stochastic waiting time until a parking spot becomes available. Bike station data is similar, except that for bikes we have two prediction files, one for predicting parking availability, and one for predicting bike availability.

4 The Policies

Traditional journey plans are a totally ordered sequence of actions, e.g., first take bus 12 and then bus 5A. In contrast, when dealing with stochasticity it is intrinsic to allow considering multiple options.

The two journey planning engines in use generate different types of policies. In the Frequency Planner's policies, the notion of a "state" boils down to location information. When multiple options are present in a state, the assumption is that the traveler will follow the first one available (e.g., take the first bus on either route 12 or route 14A). In that respect, all options have the same priority. On the other hand, in DIJA policies, a state depends on more factors, including the time and the history. For example, reaching the same location, at the same time, in two different ways can impact factors such as the amount of walking performed so far. This further impacts how much walking is acceptable in the rest of the journey, to respect a user-specified max amount. Options available in a state are prioritized and they can specify a time information (e.g., attempt to take bus 10 expected to arrive at 14:45; if missed it, walk to stop 7073).

Both types of policies have their own advantages and disadvantages. Rather than comparing them, our goal is to define a generic policy format that is sufficiently versatile to represent both types of policies. For practical reasons, we want our policies to be easy to read and understand, and to follow an established data format such as json.

A policy is an unordered list of *statements*. A statement is a collection of fields that describe a user action (specifically, following a journey leg), a specific context ("state") when the action can be considered, group information (optional) and a priority (optional).

Fields describing the user action include the transport mode, the target state of the action and, depending on the mode, other relevant information. For instance, for a bus trip, this includes the route id, the trip id (optional), the expected departure time (optional), the expected arrival time (optional), and the headsign (optional). As mentioned earlier, we allow a flexible notion of a state, which is why part of the statement fields describing a state are optional. State fields include the location, a time interval (optional) and a

state id that can uniquely identify a state in the policy graph (optional).

For instance, the following statement, written in json, specifies that at stop 89 we should take bus 12 to stop 32, and from there we should take bus 5A to stop 64.

```
"policy" : [    {"loc_type" : "stop",
                 "loc_id" : "89",
                 "to_loc_type" : "stop",
                 "to_loc_id" : "32",
                 "transport_mode" : "bus",
                 "route_id" : "12"},

                {"loc_type" : "stop",
                 "loc_id" : "32",
                 "to_loc_type" : "stop",
                 "to_loc_id" : "64"
                 "transport_mode" : "bus",
                 "route_id" : "5A"}]
```

Having different location types simplifies the merging of different geographic data sources without merging the id spaces. For example, a location of type stop would refer to a public transport stop in the corresponding GTFS files.

Now consider the case that this policy additionally contains the following statement, which is a copy of the second statement, but with a different route id:

```
"loc_type" : "stop",
"loc_id" : "32",
"to_loc_type" : "stop",
"to_loc_id" : "64"
"transport_mode" : "bus",
"route_id" : "3"
```

As no priorities are assigned to the two statements applicable to stop 32, the semantics are that the user will take the first bus on any of these routes. This is particularly useful when GTFS data provide the frequency of a service, but not specific arrival and departure times. Providing different options via statements can decrease the waiting time.

Such simple tree-like structures are powerful, but sometimes not expressive enough. This is why we allow optional fields to specify timing information, priorities, history (through unique state ids) and groups of statements.

DIJA (Botea et al. 2013) uses a more rigurous timing regime and priorities. Consider a bus route, say route 10, where buses arrive every 30 minutes. An optimal plan might be as follows: try to take the bus on route 10 arriving at 14:45; if missed, walk to stop 7073, and take another bus from there. In other words, waiting for the bus 10 arriving at 15:15 would be suboptimal. Making use of time intervals and priorities, this plan can be encoded as follows: have a statement B for the bus arriving at 14:45, with a high priority assigned, and with a time interval that, in our example, accounts for the uncertainty in the actual arrival time (e.g., 14:40 to 14:50). Have another statement W for the walking leg, with a lower priority.

Statements can be grouped together using an optional group id field. This allows to treat a bunch of nearby stops as one single location, being visited by the union of all buses passing through these stops. Consequently, all statements with the same group id are executed as if they belong to the same location.

5 Integration with IBM IIT and IBM IOC

This section focuses on integrating our system with two commercial products, IIT and IOC.

IBM Intelligent Transportation

IIT is a product for the management and prediction of traffic in cities. It can collect and process data coming from different sources, such as public transport agencies and vehicles, traffic lights and several types of sensors deployed in a city.

Based on raw data such as GPS data, IIT can dynamically update ETAs for already started trips. Furthermore, actual arrival/departure times can be stored in a database as historical data. These are two of the most relevant features to Docit.

In Docit, better estimates of ETAs can be used to compute plans from scratch, to check the validity of an existing plan, and to replan. As the time window of dynamically computed ETAs is limited, covering vehicle trips started but not completed, this time window overlaps particularly well with the second (plan validity checking) and the third (replanning) tasks. The availability of acurrate ETAs, updated in real time, could be key to the effectiveness of such tasks.

Dynamic data are stored in several "third-party" databases with a proprietary schema, that differ from the format required by Docit, presented in Section 3. The Snapshot Aggregator contains a few modules to fetch updates on dynamic data. The Public Transport Updater (PTU) queries IIT databases for public transport ETAs. The Bike Station Updater (BSU) and the Car Park Updater (CPU) obtain updates on bike stations and car parking lots, respectively. Even though the bike and parking databases are technically not part of the IIT product, we discuss the BSU and the CPU together with the PTU, for a simpler paper structuring. In the network snapshot, dynamic updates are combined with more static parts of the network, such as the list of all stops, which are stored as files on the disk.

The PTU includes a collection of 5 SQL queries. Each query involves between 4 and 7 tables in a relational database, to a total of 13 tables. It starts by retrieving all routes and stops available in the data, which are needed to eventually retrieve all public transport trips currently running. These running trips are integrated into the snapshot, combined with static data, such as trips planned to run in the future. Creating the running trip data in our format is a multi-thread process, as the data are independent from one trip to another, and that there can be potentially many trips and stops per trip.

The Bike Station Updater (BSU) queries the bike database periodically – every 10 minutes in the current implementation – to retrieve the number of bikes and free parking slots available at each station. Given this information, the BSU uses the prediction algorithm presented in Section 3 to generate the prediction of availability and the expected waiting times, for the subsequent 24 hours, with a granularity of 5 minutes. The Car Park Updater works similarly.

IBM Intelligent Operations Center

The IOC provides an executive dashboard to city operators. Its functionality spans across multiple aspects of city management, including public safety, transportation, water, social services and emergency management.

While broad on their own, these functions lack awareness information extracted from the direct experiences of travellers. At the same time, IOC allows to integrate additional functionality into its dashboard relatively easitly. Our objective is to integrate such new awareness data. First, we developped a visualisation of active journeys registered in the system, updated in real time. Secondly, we provide a visualisation of key performance indicators based on historical data, such as hotspots of frequently missed connections. Finally, the console can be used to tune system parameters.

Active journey visualization requires the creation and the configuration of two data sources via the administrative console of IBM IOC. The first one stores, in near real-time,[4] the status of the active user sessions, including information such as the current location of the user. The second data source stores the trajectory of the executed part of the journey plan policy. Their contents is updated regularly, with data from the Active Session Manager, taking advantage of the REST (Fielding et al. 1999) API provided in IOC (IBM 2013). Each update is performed with a POST (Fielding et al. 1999) request to the Data Injection Service, which is a service utilized for adding, updating and cancelling records for a data source. The frequency of the updates is a parameter specified in the configuration of Docit. Once the datasets are regularly updated, IOC automatically displays their contents in a map view created for this purpose.

Other functions we created include a system configuration view and the ability to visualise statistics about current and past trips. We extended the actual web interface of IBM IOC with an additional web application providing an interface to internal Docit data. Each different function was wrapped in an independent web-clip (Abdel-Hafez et al. 2014). In turn, each web-clip was integrated in the layout of the web portal which provides IBM IOC.

6 Use Case

In this section we present a use case that illustrates the way our system functions. For the reviewing, the use case is also provided as a *video*, uploaded as *supplementary material*.

We tested our system on data from a European city. The public transport network includes buses and trams, to a total of 36 routes, 1,297 stops, and 5,985 trips per day. There are 81 bike stations and 12 car parking lots. The road network data that we use has about 150,000 nodes and 160,000 links.

An instance of IIT has recently been installed, but its population with data is still in progress. This is a task beyond the control of the authors of this paper. For this reason, we have tested the system with static data and simulated dynamic data. For each arrival time of public transport trips, we have added an uncertainty represented as a Normal distribution, of mean $\mu = 0$ and variance $\sigma^2 = 6400$, cut to a 99.7% confidence interval. This corresponds roughly to a ± 4 minute variation in the arrival times.

In the use case at hand, a user uses his mobile app to submit a journey plan request from JEAN GALVIN to STADE. The request goes to the server, which returns a journey plan.

During the journey, the mobile app informs the server about the location of the traveller along the plan. In our implementation of the mobile app, everytime a new leg is started, the user explicitly pushes a button that indicates the step taken (among one or several steps possible in a given state) and the time. In principle, this could be performed automatically, with transport mode detection and vehicle detection algorithms (Stenneth et al. 2011). Nevertheless, this is a feature of a mobile app, not a feature of the server.

The first plan step in the example is to walk to stop PERGOLA. When the user indicates that the first leg of the plan has begun, he becomes an active user, and his trajectory is visible on the corresponding view (i.e., the one showing all active journeys) of the network operator console. Then, two alternatives are provided: take a trip to LATTES CENTRE, with the expected departure at 13:43, or take a trip to GARE SAINT-ROCH, with 13:50 being the expected departure time. The first, preferred connection is missed, and the traveller indicates that he follows the second alternative.

As travelling along the second leg progresses, we simulate an incident on a tram route that will interrupt any tram service on that route for one hour. The simulation is implemented by removing all the corresponding tram trips from the next snapshot loaded into the system. This impacts the plan at hand, which was meant to continue on tram route L4.

The system notifies the traveller as soon as the tram delays caused by the incident are reflected into a freshly computed network snapshot. Without this function, the traveller would find out about the tram service disruption only when arriving at the connection point. That could be too late, however. Clearly, the sooner replanning is performed, the more options are available, enlarging the space of available plans and thus improving the quality of the best available plans.

7 Conclusion

Existing systems for multi-modal journey planning work under deterministic assumptions. We have presented Docit, an uncertainty-aware system for multi-modal journey advising. We have described the system main functions, directed to two categories of users, namely travellers and network operators. Integration aspects that we have faced in this work include data aggregation, integration of diverse uncertainty-aware planning engines, and integration with commercial products. Our system has been integrated the IBM IIT and IOC products, and with databases fed in real time with transportation network data from a European city.

In future work, we plan to extend the functionality of the system into a combined travel and activity planner (with tourism being an application domain) capable of reasoning about uncertainty in the data. In addition, we will consider improving the user interface in our client apps. For instance, visualise in an intuitive, easy-to-grasp format the risk (e.g., the distribution of the arrival times) associated with a stochastic journey plan.

[4] Depending on the frequency of the updates received from mobile apps. See an example in the next section.

References

Abdel-Hafez, H.; Balakrishnan, S.; Caffrey, J.; Francellino, E.; Mishra, S.; Nascimento, T.; Ravichandran, J.; Scott, C.; and Vlasov, N. 2014. *IBM Intelligent Operations Center v1.6 Programming Guide*.

Botea, A.; Nikolova, E.; and Berlingerio, M. 2013. Multi-modal journey planning in the presence of uncertainty. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS-13*.

Chen, B.; Pinelli, F.; Sinn, M.; Botea, A.; and Calabrese, F. 2013. Uncertainty in urban mobility: Predicting waiting times for shared bicycles and parking lots. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, 53–58.

Fielding, R. T.; Nielsen, H. F.; and Berners-Lee, T. 1999. Internet draft: Hypertext transfer protocol - http/1.1. Technical report, W3C.

IBM. 2013. *IBM Intelligent Operations Center v1.6 REST APIs*.

Nilsson, N. J. 1968. Searching problem-solving and game-playing trees for minimal cost solutions. In *IFIP Congress (2)*, 1556–1562.

Nonner, T., and Laumanns, M. 2014. Shortest path with alternatives for uniform arrival times: Algorithms and experiments. In *Proceedings of ATMOS'14*.

Nonner, T. 2012. Polynomial-time approximation schemes for shortest path with alternatives. In Epstein, L., and Ferragina, P., eds., *ESA*, volume 7501 of *Lecture Notes in Computer Science*, 755–765. Springer.

Stenneth, L.; Wolfson, O.; Yu, P. S.; and Xu, B. 2011. Transportation mode detection using mobile phones and gis information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 54–63. ACM.