Smart Trip Alternatives for the Curious

Damien Graux, Pierre Genevès, Nabil Layaïda

Smart Trip Alternatives for the Curious

Damien Graux[1], Pierre Genevès[2], and Nabil Layaïda[1]

[1]Inria, {damien.graux,nabil.layaida}@inria.fr
[2]Cnrs, pierre.geneves@cnrs.fr

July 5, 2016

Abstract

When searching for flights, current systems often suggest routes involving waiting times at stopovers or even overnight stays between connecting flights. There might exist alternative routes which are more attractive from a touristic perspective because their duration is not necessarily much longer while offering just enough time in an appropriate place: not too short nor too long. Choosing among such alternatives requires additional planning efforts to make sure that e.g. points of interest can conveniently be reached in the allowed time frame.

We present a system that automatically computes smart trip alternatives between two cities in the world. To do so, it searches points of interest in large semantic datasets taking into account the set of accessible areas around each possible layover. It can then elect two feasible alternatives while displaying their differences with respect to the default trip.

1 Introduction

In trip planning, it is very common to query for flight combinations according to criteria such as shortest total duration, or cheapest combination for instance. Resulting routes often include inescapable waiting times at airports between connecting flights. Instead, other trip alternatives might reveal much more interesting, such as those setting an appropriate time between connections to allow for a specific activity that leverages the local environment. For instance, when travelling from Lyon to Singapore, the shortest duration criterion yields a stopover in Dubai with a waiting time of 3 hours. This might represent significant waiting time, while being too short for an activity outside of the airport. Instead, it might be more interesting to slightly defer the connection (by e.g. 2 hours) and obtain enough time to enjoy Dubai's city on the way to Singapore; or to save a hotel night at destination when the initial connection arrives in the late evening at the profit of daytime spent at an alternate stopover such

as Frankfurt. Choosing among such alternatives requires additional planning efforts to make sure that e.g. points of interest can effectively and conveniently be reached in the allowed time frame, attractions of interest are open, etc. This additional effort is particularly significant when the user is not aware of local possibilities at all viable stopovers.

We introduce a system that computes and suggests smart trip alternatives automatically, given any two origin and destination cities in the world. Our system explores large universes of semantically-checked possibilities in the set of all viable layovers, from which it automatically selects a few relevant options. Our system finally suggests two feasible smart trip alternatives while displaying their differences with respect to the default trip with e.g. shortest duration. Thus our system does not require any additional user input when compared to a system such as Google Maps [1] or Rome2Rio [2].

Our system leverages the increasing availability of open city transportation data (in e.g. GTFS [9]), and combines them with flight information as well with external data sources for the selection of e.g. particularly remarkable points of interests. In Section 2, we present how we designed our system around a scalable infrastructure for supporting the mass of worldwide GTFS information, how we leverage various data sources in heterogeneous formats (e.g. RDF, JSON, XML, GTFS, etc.) for semantic enrichment of information, and how we encode constraints and heuristics for the efficient selection of smart trip options. In Section 3 we illustrate the use of our novel system in a real-world setting before reviewing related works and concluding in Section 4.

2 Overall System Architecture

The global architecture of our system is shown in Figure 1. It consists of a lightweight client-side part in which users indicate a city of origin and a city of destination and which also displays results, and a backend part with an entry point called master. As shown in Figure 1, the master executes three different processes A,B and C. Process A corresponds to a usual flight finder: it returns trips sorted by simple criteria such as the number of connections and the transit time (by default). Process C queries the Open Street Map [10] tiles servers to fetch cartographic data for drawing resulting routes on a map. Processes A and C basically correspond to what can be found in common flight finding applications. The novelty of our idea and our system resides in process B, which is in charge of computing recommendations by reasoning on enriched data. This process performs the semantic searches, verifications, and filters that finally yields suggested smart trip options.

In the backend we distinguish datasets according to their sizes. For performance reasons, when datasets fit in main-memory of a single machine, we make sure that computations from the performances of in-memory engines. This is the case for the flight database and the restaurant databases that fit on a single node whereas city transit data is distributed across a cluster of nodes. Indeed,

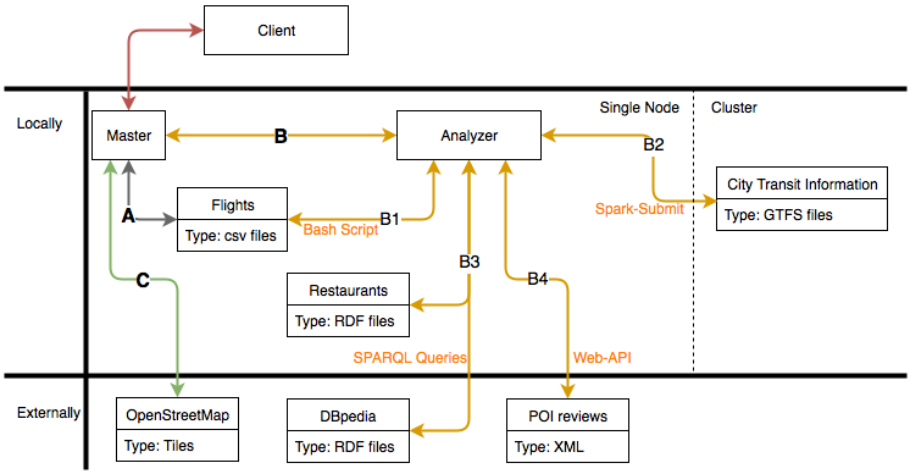

Figure 1: Overall Architecture.

the sizes of these databases are not of the same order of magnitude. For example, for the Los Angeles (California) city area and its main airport, flight and restaurant data represent 2Mb, whereas the size of public transportation data for the same area is closed to 20Gb (due to at least seventy million direct paths between all regional stations).

Our system store city transit information using the General Transit Feed Specification (GTFS) [9] datasets, which consist of several CSV files providing routes, schedules, stations and stop times for instance.

For the purpose of scalability with the increasing amount of GTFS data made available by transit agencies, we designed our GTFS store on top of Apache Spark[1]. We store GTFS data in terms of resilient distributed datasets (RDDs) [12] over which we issue queries with Spark's dataframe API.

To obtain smart trip alternatives for a transit between two airports A_1 and A_2, process B (see Figure 1) performs reasoning on aggregated data coming from various sources thanks to the four sub-processes B1, B2, B3, and B4 (shown on Figure 1).

First of all, the analyzer queries the flight finder (B1) to have all possible paths (without cycles) taking off from A_1 and landing at A_2. Then, it applies filters to this set of paths according to two default (customizable) usecases: (1) it keeps paths having at least a three-to-five hour connection; (2) it only considers paths having at least a connection longer than eight hours. Moreover, it always tries to avoid connections requiring to spend one night in a hotel somewhere on Earth and rather promotes night in aircrafts, by default.

Knowing the possible connections, the analyzer asks the GTFS store (B2)

[1] http://spark.apache.org/

to find among the city transit datasets all the accessible areas from each intermediate airport. This concept of accessibility depends on the usecase *i.e.* the GTFS store only considers areas from which one can go and return in less than M minutes, where M is equal to the minimum between one quarter of the connection time and 2 hours. For instance, if the connection time equals 3 hours we do not select areas located more than 45 minutes round trip; and if this connection lasts more than 8 hours, public transport transit times are limited correspondly (e.g. to 2 hours).

A set of accessible stations (using public transport) is hence available for each possible connection. To enrich user experience (B3), the analyzer seeks points of interest (POIs) in these areas. It uses DBpedia [4] as semantic data provider since DBpedia extracts factual information from Wikipedia and converts it into RDF. Practically, the analyzer sets up SPARQL queries [3]. Figure 2 presents a typical SPARQL query that might be generated.

We have implemented various strategies to limit the number of results. First, the SPARQL query selects elements of type dbpedia:Place and returns for each place its name, a short abstract, and a picture if possible using the OPTIONAL SPARQL feature. In the same time, a language filter is applied on names and abstracts with FILTER(lang...). Finally, it keeps only the five closest POIs (LIMIT and ORDER BY) around the station (X and Y as GPS coordinates) within a radius RAD. Since several treated stations might be close, we define the different RAD not to have overlapping areas; we thus guarantee that each POI belongs to only one accessible station. Meantime, the analyzer queries our local restaurant RDF dataset to suggest places to eat while users do some sightseeing.

Then, in (B4) the analyzer fetches ranks and reviews of other users concerning all accessible POIs. Such an operation aims at adding human opinion within the processing chain: having ranks allows an *a priori* POI classification. To refine the best trip option, the analyzer considers several parameters: the average rank of an area, the effective time that is spent in this area, and even the length of the various abstracts. All these considerations are used to obtain an overall score for each area; the analyzer can thereby choose among the best retained ones using a (customizable) score function.

Finally, the master joins obtained GPS coordinates with OSM map tiles and then sends back to the client the three best results: first the fastest trip minimizing the number of connections and the overall trip duration, second an interesting trip alternative taking advantage of a connection lasting three to five hours, third another interesting alternative when more than eight hours can be spent somewhere.

3 Typical System Usage

The typical scenario consists in using our processing pipeline in order to obtain smart trip alternatives using various data sources: GTFS schedules, OSM tiles

```
SELECT ?name ?abstract ?picture
WHERE {
  ?c  rdf:type           dbpedia:Place .
  ?c  dbpedia:abstract   ?abstract .
  ?c  rdfs:label         ?name .
  ?c  geo:long           ?long .
  ?c  geo:lat            ?lat .
  OPTIONAL {
    ?c  dbpedia:thumbnail  ?picture .
  }
  BIND(((xsd:double(?lat)-X)^2)+
       ((xsd:double(?long)-Y)^2)
      ) AS ?dist
  FILTER(lang(?name) = "LANG")
  FILTER(lang(?abstract) = "LANG")
  FILTER(?dist < RAD)
}
ORDER BY ASC (?dist)
LIMIT 5
```

Figure 2: SPARQL query extracting from dbpedia the 5 closest POIs in language LANG located around a point whose GPS coordinates are (X,Y) within a radius of RAD.

and DBpedia RDF data. One interest of such a tool relies on the fact that users can find alternatives using real data *e.g.* the scheduling grids are the ones used each day by official transit agencies and semantic data comes from DBpedia. For instance it is possible to review suggestions of trip alternatives to come to the conference.

3.1 Sample of Real Datasets

We preloaded GTFS datasets of various transport agencies around the world. We extracted most of them from the GTFS data exchange platform[2]. In terms of disk footprint, they represent more than 50Gb on the cluster. Table 1 summarizes information of some datasets, *e.g.* numbers of routes or direct pairs or also stops.

3.2 Step-By-Step Usage Example

System users can search for trip alternatives passing as argument two airports and the two allowed time lapses for connections. For instance, from Paris in France (CDG airport) to Honolulu in Hawaii USA (HNL airport) with the de-

[2]http://www.gtfs-data-exchange.com/

Datasets	Stops	Routes	Direct Pairs	Loaded Size (Gb)
Los Angeles Ca	14 992	148	74 570 202	19,8
San Francisco Ca	4 577	85	17 358 866	7
New York City	17 923	1 317	146 231 113	44
Paris, France	3 204	34	2 534 273	1

Table 1: Information of some datasets.

fault usecases, the pipeline might propose at first (process A in Figure 1) to pass through San Francisco Ca since it is the fastest trip available, all connections via Los Angeles Ca or via other airports are longer:

== The 3 Fastest Trips ==
CDG (10:30--12:55) SFO (13:30--17:20) HNL
CDG (11:30--14:30) LAX (14:40--18:22) HNL
CDG (13:30--16:15) LAX (16:45--20:35) HNL

After acting as a conventional "fastest trip finder", the application computes smart alternative trips using the various tools previously presented (Section 2) and grouped in the process B in Figure 1. First of all, it searches trips having a three-to-five hour connection; considering the example introduced above (CDG→HNL), only 14 options are possible, and 4 airports can be used as connection: San Francisco Ca (SFO), Los Angeles Ca (LAX), Seattle Wa (SEA) and Tokyo Japan (NRT). These trips sorted in descending order of connection time are:

CDG (21:20--17:05) NRT (22:00--09:35) HNL
CDG (11:30--14:30) LAX (18:45--22:33) HNL
CDG (13:30--15:13) SEA (19:25--23:58) HNL
CDG (09:10--12:15) SFO (16:05--19:42) HNL

For each possible stopover, the GTFS store is used to identify all the viable accessible areas from the airports. Then the application looks for closest POIs around these areas. In the CDG→HNL case, it considers for instance the Naritasan Shinsho-ji Temple (20 minutes far from NRT) or also Venice Beach

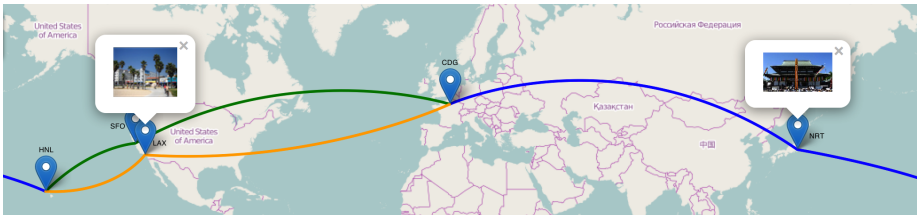

Figure 3: Application Screenshot (CDG→HNL).

(40 minutes far from LAX). At the end, taking into account the effective time available in each possible sites and the ranking averages of various areas, our application suggests a stopover in Narita.

Then, a second alternative is presented to attendees. However, this option which should take advantage of a long connection time (more than eight hour long) does not appear to be valuable in the case CDG→HNL. Actually, each trip involves an overnight stay in a hotel, even if new intermediate destinations are available *e.g.* Atlanta Ga, Dallas Tx, Chicago Il, or Vancouver Canada. Rather than returning an empty result, the application uses the list made in the previous case (with a three-to-five hour connection) and returns the second most interesting trip *i.e.* passing through Los Angeles.

Finally, in addition to the fast Paris-Honolulu via San Francisco, two alternative journeys are proposed (Figure 3): one via Tokyo and another one via Los Angeles.

4 Related Work and Conclusion

There exists many trip planning systems such as Google Maps [1] and Rome2Rio [2] for example. These systems allow one to easily obtain routes that satisfy simple criteria such as shortest path, shortest duration, cheapest price, and combinations of them. Compared to these systems, we bring an additional semantic layer that allows our system to suggest smart alternatives, e.g. alternatives that do not necessarily satisfy the initial criteria entered by the user, but that will be preferred in the end.

Using DBpedia [4] as a POI provider in a tourism context has been proposed by [5, 7]. We used DBpedia similarly in the more specific case of journey planning.

Closest to our approach are the works on automatic construction of travel itineraries [6, 8] and interactive itinerary planning [11]. These approaches typically look for feasible itineraries given a particular location and time budget. Specifically, the approach in [11] introduces an interactive planning process that starts with a user providing a time budget and a starting point of the itinerary (usually corresponding to the hotel where the user is staying). The system progressively suggests a touristic itinerary depending on successive user feedbacks until the user chooses a specific tour.

Compared to these approaches, we notably leverage the use of GTFS big data [9] for checking feasibility of the itinerary by public transportation. Furthermore, our generic architecture might benefit from the developments in [7, 11] for generating even more alternatives. Indeed our processes B3 and B4 shown on Figure 1 might also include such additional systems to improve the set of relevant alternatives.

Last but not least, an advantage of our system compared with [8, 11] is to provide alternatives at booking time. The user becomes active in the layover decision process deciding how and where to spend its time budget.

References

[1] Google-Maps. https://www.google.fr/maps.

[2] Rome2Rio. http://www.rome2rio.com/.

[3] SPARQL 1.1 overview, March 2013. http://www.w3.org/TR/sparql11-overview/.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *DBpedia: A nucleus for a web of open data*. Springer, 2007.

[5] A. E. Cano, G. Burel, A.-S. Dadzie, and F. Ciravegna. Topica: A tool for visualising emerging semantics of pois based on social awareness streams. In *10th Int. Semantic Web Conf (ISWC2011)(Demo Track)*, 2011.

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

[7] S. Cresci, A. D'Errico, D. Gazzé, A. Lo Duca, A. Marchetti, and M. Tesconi. Towards a DBpedia of tourism: the case of tourpedia. In *Proceedings of the 2014 International Conference on Semantic Web-Poster and Demo Track, ISWC2014*, pages 129–132, 2014.

[8] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, pages 35–44. ACM, 2010.

[9] Google. GTFS definition, September 2006. https://developers.google.com/transit/gtfs/.

[10] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[11] S. B. Roy, G. Das, S. Amer-Yahia, and C. Yu. Interactive itinerary planning. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 15–26. IEEE, 2011.

[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI*, 2012.