Smart Trip Alternatives for the Curious

Damien Graux, Pierre Genevès, and Nabil Layaïda

Inria, Cnrs, lig and Univ. Grenoble Alpes, France

Abstract. When searching for flights, current systems often suggest routes involving waiting times at stopovers. There might exist alternative routes which are more attractive from a touristic perspective because their duration is not necessarily much longer while offering enough time in an appropriate place. 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 any two cities. To do so, it searches points of interest in large semantic datasets considering the set of accessible areas around each possible layover. It then elects feasible alternatives and displays 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. Instead, it might be more interesting to slightly defer the connection and obtain enough time to enjoy Dubai's city on the way to Singapore, for instance. 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 and tolerated connection times. 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 systems such as Google Maps or Rome2Rio[1].

[1] http://www.google.com/maps & http://rome2rio.com

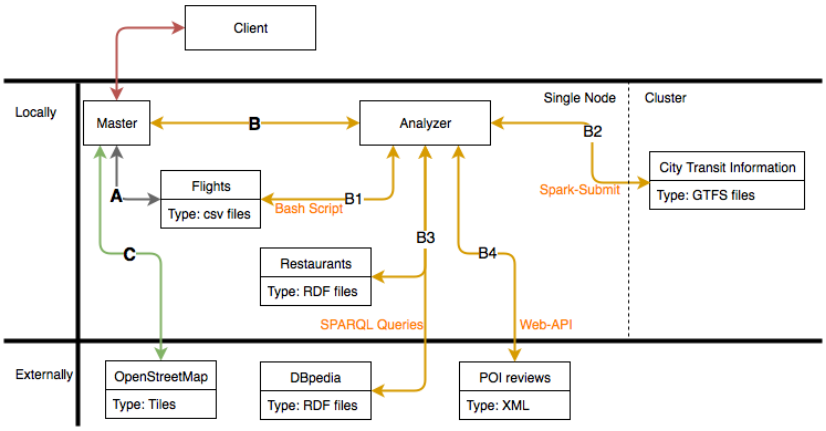

Fig. 1. Overall Architecture.

Our system leverages the increasing availability of open city transportation data (in *e.g.* the General Transit Feed Specification (GTFS) [3]), 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 (*i.e.* several CSV files providing routes, schedules, stations stop times...), 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 (OSM) [5] 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.

Design for scalability. In the backend we distinguish datasets according to their sizes. For performance reasons, when datasets fit in main-memory of a single

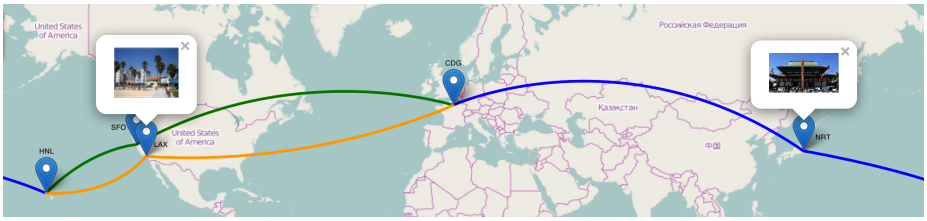

Fig. 2. Application Screenshot: Paris to Honolulu (CDG→HNL).

machine, we use in-memory engines (for *e.g.* the flight database) whereas city transit datasets (expressed using GTFS [3]) and large RDF datasets are distributed across a cluster of nodes. Specifically, we use a GTFS store implemented on top of the Apache Spark framework[2] for the purpose of supporting a large number of GTFS datasets of moderate size[3]. When RDF datasets used in the data enrichment process (B3) are large, we rely on our SPARQLGX implementation introduced in [4] for efficiently evaluating SPARQL queries on distributed RDF datasets.

To obtain smart trip alternatives for a transit between two airports A_1 and A_2, process B performs reasoning on aggregated data coming from various sources thanks to the four sub-processes B1, B2, B3, and B4 (Figure 1). The analyzer first queries the flight finder (B1) to gather 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 customizable usecases: (1) it keeps paths having at most a X-hour connection; (2) it only considers paths having at least a connection longer than Y hours; where X and Y are user-defined. 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) 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. A set of accessible stations (using public transport) is hence available for each possible connection. To enrich user experience, the analyzer first seeks points of interest (POIs) in these areas using SPARQL queries evaluated on DBpedia[4], and then further queries other RDF databases for semantic enrichment (B3), *e.g.* with local restaurants. Then, in (B4) the analyzer fetches ranks and reviews of other users concerning all accessible POIs. 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.

[2] http://spark.apache.org
[3] For example, GTFS data for the Los Angeles city area represent 20GB (due to seventy million direct paths between all regional stations).
[4] http://dbpedia.org

3 Demonstration Scenario

The typical scenario consists in using our processing pipeline in order to obtain smart trip alternatives using various data sources: GTFS schedules, OSM tiles 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. Users can search for trip alternatives passing as argument two airports and two allowed time lapses for connections. For instance, from Paris in France (CDG) to Honolulu in Hawaii USA (HNL) allowing 3 to 5 hours and more than 8 hours, the pipeline (Figure 1) might propose at first to pass through San Francisco CA (as a conventional "fastest trip finder") since it is the fastest trip available. Then, it considers *e.g.* the Naritasan Shinsho-ji Temple (20 min. far from Tokyo) and also Venice Beach (40 min. far from Los Angeles).

4 Related Work and Conclusion

There exists many trip planning systems such as Google Maps and Rome2Rio. These systems allow to 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. Closest to our approach are the works on automatic construction of travel itineraries [1, 2] and interactive itinerary planning [6]. Compared to these approaches, we notably leverage the use of GTFS big data [3] for checking feasibility of the itinerary by public transportation. Furthermore, an advantage of our system compared with [2, 6] 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. Chen, G., Wu, S., Zhou, J., Tung, A.K.: Automatic itinerary planning for traveling services. TKDE 26(3), 514–527 (2014)
2. De Choudhury, M., Feldman, M., Amer-Yahia, S., Golbandi, N., Lempel, R., Yu, C.: Automatic construction of travel itineraries using social breadcrumbs. In: ACM – Hypertext and Hypermedia. pp. 35–44. ACM (2010)
3. Google: GTFS (September 2006), https://developers.google.com/transit/gtfs/
4. Graux, D., Jachiet, L., Genevès, P., Layaïda, N.: SPARQLGX: A distributed RDF store mapping SPARQL to Spark. ISWC (2016)
5. Haklay, M., Weber, P.: Openstreetmap: User-generated street maps. Pervasive Computing, IEEE 7(4), 12–18 (2008)
6. Roy, S.B., Das, G., Amer-Yahia, S., Yu, C.: Interactive itinerary planning. In: ICDE. pp. 15–26. IEEE (2011)