# Scaling up the Mining of Semantically-enriched Trajectories: TripBuilder at the World Level

Igo Brilhante[1], Jose Antonio Macedo[1],
Franco Maria Nardini[2], Raffaele Perego[2], Chiara Renso[2]
[1] Federal University of Ceará, Brazil – [2] ISTI-CNR, Italy

**Abstract.** TripBuilder is an unsupervised system helping tourists to build their own personalized sightseeing tour [1, 3, 2]. Given a target touristic city, the time available for the visit, and the tourist's profile, TripBuilder provides a time-budgeted tour that maximizes tourist's interests and takes into account both the time needed to enjoy the attractions and to move from one Point of Interest (PoI) to the next one. The knowledge base feeding the sightseeing tour generation algorithm of TripBuilder is entirely mined from publicly available sources, namely, Wikipedia, Flickr and Google Maps. This paper introduces a scalable and robust Cloud architecture (combining both stream and batch processing) to download the data from the heterogeneous sources and build a huge TripBuilder knowledge base covering most popular cities worldwide.

## 1 Introduction

The generation of the TripBuilder knowledge base is a unsupervised process that involves several steps:

**PoIs**. The first step is to identify the set of points of interest (PoI) in the target city. Given the bounding box $BB_{city}$ containing the city, we download all the geo-referenced Wikipedia pages falling within this region. We assume each geo-referenced Wikipedia named entity, whose geographical coordinates falls into $BB_{city}$, to be a fine-grained PoIs. For each Wikipedia PoI, we consider its descriptive label, its geographic coordinates, and the set of Wikipedia categories the PoI belongs to. By considering the set $C$ of categories associated with all the Wikipedia PoIs, we generate the normalized relevance vector of each PoI. Since a tourist in a given place can enjoy all the attractions in the surroundings, we also perform a density-based clustering to group in a single PoI sightseeing entities which are very close one to each other. At the end of this step each PoI $p \in \mathcal{P}$ of the city is enriched with its geographic coordinates, a name and a description, and a *relevance vector*, $\boldsymbol{v_p} \in [0,1]^{|C|}$, measuring its normalized relevance of $p$ w.r.t the categories $C$.

**Users and PoI histories**. As second step we need a method for collecting tourists and their long-term itineraries crossing the discovered PoIs. We query Flickr to retrieve the metadata (*user id, timestamp, tags, geographic coordinates*, etc.) of geo-referenced photos taken in the given area $BB_{city}$. The assumption we are making is that photo albums made by Flickr users implicitly represent sightseeing itineraries within the city. This process thus collects a large set of geo-tagged photo albums taken by different users within $BB_{city}$. We discard

photo albums containing only one photo and we spatially match the photos with the set of PoIs previously collected. Moreover, we consider the timestamps associated with the first and last photos taken by each user in a given PoI to estimate the average PoI visiting time $\rho(p)$. The popularity $pop(p)$ of each PoI $p$ is computed instead by normalizing the number of distinct users that shot it in at least one photo. The above process allows us to generate the set of users, their PoI history (the temporally ordered sequence of PoIs visited by a user $u$), and estimate for the popularity and visiting time of each PoI. Finally, a preference vector $\boldsymbol{v_u} \in [0,1]^{|C|}$ stating the normalized interest of $u$ for the categories in $C$ is built by summing up and normalizing the relevance vectors of all the PoIs occurring in $u$ PoI history.

**Trajectories**. In order to build the set $\mathcal{S}$ of trajectories used by TripBuilder we split users' PoI histories by cutting the ordered list where the time interval between the visit to two subsequent PoIs is greater than a given threshold $\delta$ derived by analyzing the inter-arrival time of each pair of consecutive photos taken in different PoIs. Given the distribution of probability of such inter-arrival time $P(x \leq \delta)$, we compute for each city the time threshold $\delta$ such that $P(x \leq \delta) = 0.9$.

**Traveling time estimation**. TripBuilder recommend personalized sightseeing tours fitting the time budget of the user. Therefore also the time $\tau(\cdot,\cdot)$ needed to move between consecutive PoIs in the itinerary has to be estimated. Since measuring intra-PoI moving time from the photo albums resulted to be inaccurate for less popular PoIs, we query Google Maps for the distance between the PoIs.

**User-PoI Interest**. Given a PoI $p$, its relevance vector $\boldsymbol{v_p}$, a user $u$, and the associated preference vector $\boldsymbol{v_u}$, we define the *User-PoI Interest* function as a the following function $\Gamma(p,u) : \mathcal{P} \times \mathcal{U} \rightarrow [0,1]$: $\Gamma(p,u) = \alpha \cdot sim(\boldsymbol{v_p}, \boldsymbol{v_u}) + (1-\alpha) \cdot pop(p)$ where $sim(\boldsymbol{v_p}, \boldsymbol{v_u}) = \frac{\boldsymbol{v_p} \cdot \boldsymbol{v_u}}{||\boldsymbol{v_p}|| \, ||\boldsymbol{v_u}||}$ is the cosine similarity between the user preference and the PoI relevance vectors, and $\alpha \in [0,1]$ is a parameter controlling how much user preference and popularity of PoIs have to be taken into account.

**Sightseeing Tour Generation**. Once the knowledge base for a given city is available, TripBuilder addresses the problem of planning the visit to the city as a two-step process detailed in [3].

## The TripBuilder System

The architecture of the TripBuilder system involves four different layers:

**Stream Layer with Apache Storm**. This layer is composed of two different modules that retrieve the relevant information from Flickr and Wikipedia by receiving city bounding boxes as a stream. In particular, each item of the stream is used by *Photo Discovery* to query Flickr to retrieve the metadata (user id, timestamp, tags, geographic coordinates, etc.) of photo albums, i.e., sequences of photos taken in the given geographic area. An important assumption we are

doing is that photo albums implicitly represent sightseeing itineraries within a city. To strengthen the accuracy of our method, this module retrieves only the photos having the highest geo-referencing precision. This process thus collects a large set of geo-tagged photo albums taken by different users in the given geographic area. The second module, *Wikipedia PoI Discovery*, collects PoIs from Wikipedia. In particular, we assume each geo-referenced Wikipedia named entity, whose geographical coordinates falls into a given area, to be a Point of Interest. For each PoI, we retrieve its descriptive label, its geographic coordinates as reported in the Wikipedia page, and the set of categories the PoI belongs to, which are reported at the bottom of the Wikipedia page. Then, photos from Flickr and PoIs from Wikipedia are matched by spatial proximity according to their coordinates. Figure 1 highlights the components on the Stream layer.

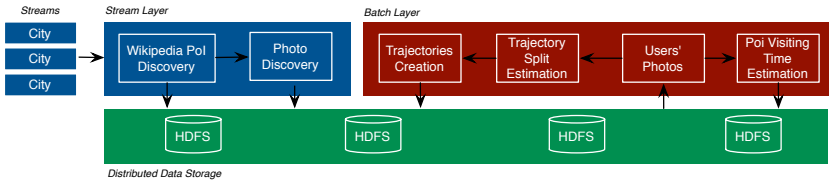


**Fig. 1.** Layers of the TripBuilder architecture.

The stream layer is built by means of Apache Storm[1], a free and open source distributed realtime computation system. Apache Storm allows to reliably process unbounded streams of data. Storm organizes the computation in a graph, called *topology*, where data flows through nodes, called *bolts*. Our stream layer is thus able to crawl Flickr and Wikipedia in a real-time fashion by receiving from an input Kafka[2] queue a given bounding box representing the target geographic area. The results of the real-time computation are stored on a distributed data storage. Figure 2 highlights the topology responsible for processing streams on TripBuilder, where *spout* nodes read data streams like city bounding boxes, PoIs, passing them through *bolt* nodes (**Wiki** and **Photo**) to discovery PoIs and photos respectively, which are stored by HDFS bolt nodes. Note that, this topology is highly scalable where *spout* and *bolt* nodes can have as many instances as needed spread across several machines.

**Batch Layer with Apache Spark**. This layer is made up of different components each one manipulating the data previously collected. It is in charge of cleaning and transforming the data by means of distributed computing frameworks like Apache Hadoop[3] and Spark[4] to speed up the data processing step. In particular, the modules here transform sequences of photos from Flickr to sequences of visited Wikipedia PoIs, i.e., trajectories, to be used in the Trip-

[1] `https://storm.apache.org`
[2] `http://kafka.apache.org/`
[3] `http://hadoop.apache.org`
[4] `http://spark.apache.org`

BUILDER module. Moreover, this step is in charge of computing popularity and other important characteristics of PoIs by considering metadata and information extracted both from Flickr and Wikipedia. We take advantage of the functional capabilities of Spark to distribute and parallelize the computation on the cloud cluster. Spark has shown to be a great tool for large-scale data processing. The data obtained are then stored on a "Distributed Data Storage" layer. This is an important point in favour of enabling the flexibility of TRIPBUILDER: different sources of information for trajectories and PoIs can be easily integrated into the system by modifying only the two lowest layers. Moreover, the approach taken allows to scale to large geographic areas as the two layers effectively exploits modern state-of-the-art technologies for distributed and parallel computation.

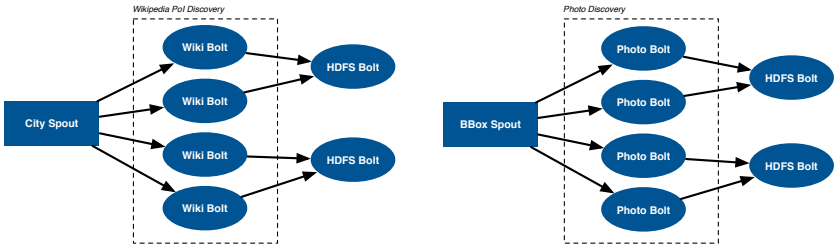


**Fig. 2.** TRIPBUILDER Storm topology.

**Distributed Data Storage**. This component is responsible for storing, querying and indexing trajectory and PoI data. It is composed by a database management system and a distributed filesystem that efficiently provides information to the "TripBuilder Engine" component and a distributed data storage to support *Stream* and *Batch* layers. The database component contains a well-defined schema to enable flexibility in integrating other data sources. Geo-spatial indexes are used for searching spatial objects, such as PoIs and tourist traces, within a given region (e.g. polygon). The system also takes advantage of indexes over PoI categories and tourist traces, both represented as arrays, to efficiently retrieve relevant PoIs to the user preferences. Moreover, the distributed filesystem is built by using the Apache Hadoop Distributed Filesystem (HDFS). We choose the HDFS technology as it is a mature solution for storing data in distributed environments. As an example, it provides effective and efficient mechanisms to deal with faults thus preventing us to avoid data loss in case of hardware problems.

## References


1. Brilhante, I.R., Macedo, J.A., Nardini, F.M., Perego, R., Renso, C.: Where shall we go today?: Planning touristic tours with tripbuilder. In: Proc CIKM'13. pp. 757–762. ACM (2013)
2. Brilhante, I.R., Macedo, J.A., Nardini, F.M., Perego, R., Renso, C.: Tripbuilder: A tool for recommending sightseeing tours. In: Proc. ECIR 2014, LNCS, vol. 8416, pp. 771–774. Springer (2014)
3. Brilhante, I.R., Macedo, J.A., Nardini, F.M., Perego, R., Renso, C.: On planning sightseeing tours with tripbuilder. IP& M 51(2), 1 – 15 (2015)