City Nexus: Discovering Pairs of Jointly-Visited Locations Based on Geo-Tagged Posts in Social Networks

Yaron Kanza
Jacobs Technion-Cornell
Innovation Institute
kanza@cornell.edu

Elad Kravi
Technion – Israel Institute of
Technology
ekravi@cs.technion.ac.il

Uri Motchan
Technion – Israel Institute of
Technology
umotchan@gmail.com

ABSTRACT

Recently, there is a rapid growth in the use of microblogs, such as Twitter, and of social networks, such as Instagram, to publish geo-tagged posts that indicate the location of the user at the time when the message is sent. This provides abundant geospatial data that can be analyzed to understand the behavior of masses of people, in particular in urban areas. Such analysis can improve and facilitate the work of urban planners and of policy makers, e.g., when deciding where to add transportation routes or public institutes. In this demonstration, we present a system that utilizes geo-tagged posts to discover places that were jointly visited by many people. We present the management and the analysis of the data, to illustrate the feasibility of the approach and to indicate new research questions in this domain.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design, Management

Keywords

Microblogs, geospatial, geo-tagged posts, urban planning

1. INTRODUCTION

Many online social networks and microblogging applications allow users to post geo-tagged messages or to specify their location by checking-in. (Facebook, Twitter, Instagram, and Foursqare are examples of such applications.) In geo-tagged messages, the location of the user at the time when the message is posted, is attached to the message. So, such massages indicate places in which users visited. Analyzing large sets of geo-tagged messages can, thus, be useful for understanding better the motion of people in cities and for discovering anomalies or significant facts about the examined urban areas.

In this project we focus on the discovery of *links* between places in a city, based on geo-tagged messages. Such messages, and in particular geo-tagged tweets, represent real user movement in the city, they can indicate recent changes and they can be easily collected from social networks. We consider two places as linked if there are many people who visited both places withing a short period of time. Since we use geo-tagged messages, our system actually discovers pairs of locations that quite a few people posted messages from both places (in near times). Discovering such places—locations that are jointly visited by many people—can be used in urban-planning tasks, in recommendation systems and may help monitoring changes in the city.

In urban planning, jointly-visited locations can indicate places where there is a need for public transportation. It can also emphasize the effect of public transportation, e.g., are there, in the examined city, many pairs of places that are connected by public transportation and are jointly visited while other similar pairs, without effective public transportation between them, are only seldom jointly visited? Far away jointly-visited places may indicate the need for facilities or services in a certain area. For example, if many people from some neighborhood visit a park far from their neighborhood, this may point out a need to add a park near or in this neighborhood.

The ability to detect jointly-visited places can be used in recommendation systems. Such systems recommend places to visit by telling a user that quite a few people who visited the location she is currently at also visited the recommended places. Note that this is similar to recommendations in online shopping ("Costumers who bought this item also bought...") except that the recommendation is of places rather than items.

A monitoring of changes can be done by running the discovery algorithms from time to time, to detect changes in the set of linked places. If two linked places cease being jointly visited or if two unlinked places suddenly start being jointly visited, this may call for an inspection. This may reveal transportation issues, or trigger an alert, e.g., if a touristic site and an hospital are suddenly jointly visited, this may divulge concealed hazards in the touristic site.

One of the difficulties in discovering jointly visited places is dealing with the masses of data and the complexity of the problem. In many similar problems of analyzing geospatial data, it is natural to cluster messages based on locations and ignore the association to the user who posted the data. Such

SIGSPATIAL '14, November 04-07 2014, Dallas/Fort Worth, TX, USA
ACM 978-1-4503-3131-9/14/11.
http://dx.doi.org/10.1145/2666310.2666378

approach works well in detection of hotspots or events. However, for finding jointly-visited places, we cannot ignore the associations of messages to users, because these associations define the linkage between places. Obviously, clustering or grouping messages by ignoring locations would fail as well, because the goal is to find linked locations based on the joined visits of many people, not just of a single individual.

To handle the problem, we developed a system that collects geo-tagged messages, stores, clusters and processes the messages efficiently to find jointly-visited locations. The system allows users to easily control various parameters in the search for linked places. In the demonstration we will present the system and its ability to effectively discover jointly visited places in different cities in the world.

2. RELATED WORK

Recently, there has been a growing interest in using location data of users for understanding better urban activities, and for improving and facilitating urban planning tasks. Ratti et al. [7] used data from cellular phones for analyzing the locations of people in the city. They generate graphic views of urban activities and present the changes in activities as a function of time and space. Ferrari et al. [4] showed how to detect hotspots in a city based on Twitter posts. Their tasks are related to our task but are different in the sense that they find crowded places and do not focus on the relationships between locations, so they can avoid using the association of posts or of other location data to the users who produced the data.

Bawa et al. [1] analyzed geo-tagged messages they had collected from Foursquare, and detected motion pattens of individuals. Differently from us, their focus is on individuals rather than on masses of people. Kling et al. [6] showed how to classify posts into urban topics related to various activities, based on the location and the content of the posts. The Livehoods project of Cranshaw et al. [2] uses geo-tagged messages from both Foursquare and Twitter to find boundaries between neighborhoods. Note that there are a few neighborhoods in a city but there can be many pairs of jointly visited locations, so the discovery of linked places requires a different approach than recognizing neighborhoods.

Some previous papers studied the problem of place recommendation (e.g. [5]) or route recommendation (e.g. [3]) based on activities in social networks, however, their approach is user-targeted and does not focus on the general discovery of jointly-visited places.

3. SYSTEM

Our system is designed as a client-server web application. The client is a web browser. Thus, we use the Google Maps API for presenting pairs of message clusters and relevant meta data, on a map. The back-end tier includes crawlers and a server that performs the computation of the clustering and the discovery of the links between clusters. The crawlers are responsible for downloading new geo-tagged messages from social networks and storing them in the database. The server clusters the messages and then finds pairs of clusters that are jointly visited by many users. The system architecture is presented in Figure 1.

By using a client-server architecture, the back-end tier can collect geo-tagged messages from social networks, compute clusters, using different parameters, and find links between clusters, on the background, without affecting the client. Next we elaborate on the components of the system.

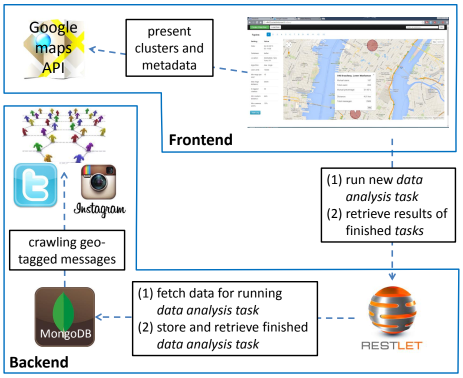

Google maps API
present clusters and metadata
Frontend
(1) run new *data analysis task*
(2) retrieve results of finished *tasks*
Instagram
crawling geo-tagged messages
MongoDB
(1) fetch data for running *data analysis task*
(2) store and retrieve finished *data analysis task*
RESTLET
Backend

Figure 1: System Architecture

3.1 The Front-end

We present now the client side of the system.

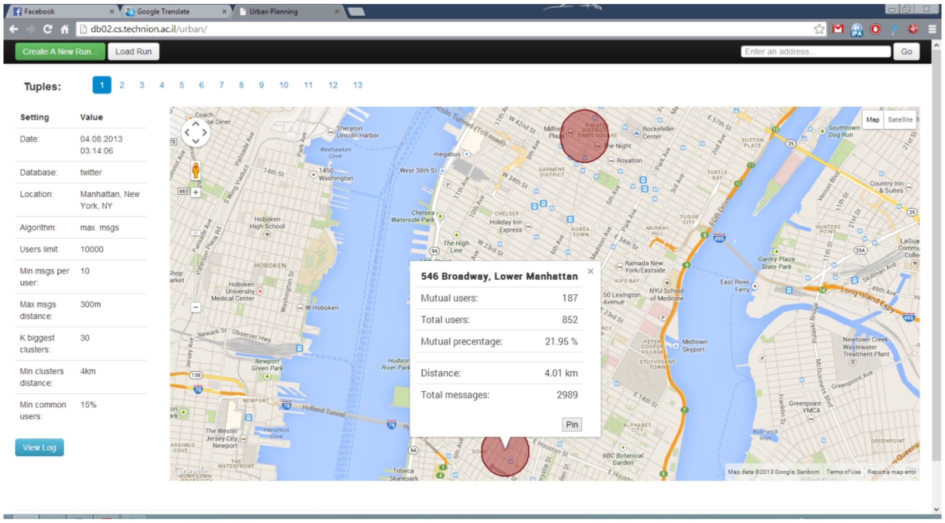
3.1.1 Issuing Data Analysis Tasks

One of the difficulties in the discovery of linked places is that it may not be clear what "place" means. In some cases a place may refer to a building and in other cases it may refer to an area, such as a park or a neighborhood. Hence, the system allows the user to insert parameters that control the computation of the clusters and of the links between them. The system allows users to insert parameters such as the minimal number of messages in a cluster, the minimal distance between clusters and the maximal distance between messages in a cluster. To control the creation of links, users can specify a threshold for the percentage of users who have a message in both clusters out for the total number of users associated with one of the clusters.

The computations required for a data analysis task may not be immediate. Hence, a data analysis task is sent as an Ajax request to the server, allowing the server to notify the client when the task is completed. Upon completion of a task, the task is stored in the database and a link to it is added to a table of complete tasks, so users can view complete tasks whenever they desire.

3.1.2 Displaying Analysis Results

Complete analysis tasks can be viewed by the user. Figure 2 presents a view of a complete task. The client page was designed using Twitter's Bootstrap CSS and Javascript package (`http://getbootstrap.com/`). At the top of the screen, the user can initiate a new data analysis task or load an existing task from the complete-tasks table. From the row below, the user may select pairs of linked places, of the currently viewed analysis task. For two clusters C_1 and C_2, the *mutual users* of these clusters are those users who have at least one message in C_1 and at least one message in C_2. The pairs are ordered by the number of mutual users of each pair. That is, the first pair is the one with the largest number of mutual users and the last pair is the one with the least number of mutual users. On the left side of the screen, the settings of the current data analysis task are presented. The pairs are presented on a map using the Google Maps API (`https://developers.google.com/maps/`).



Setting	Value
Date:	04.08.2013 03:14:06
Database:	twitter
Location:	Manhattan, New York, NY
Algorithm:	max. msgs
Users limit:	10000
Min msgs per user:	10
Max msgs distance:	300m
K biggest clusters:	30
Min clusters distance:	4km
Min common users:	15%

546 Broadway, Lower Manhattan

Mutual users:	187
Total users:	852
Mutual precentage:	21.95 %
Distance:	4.01 km
Total messages:	2989

Figure 2: Screenshot of our System

The view in Figure 2 includes a pair of clusters—each cluster is presented as a red circle. The system presents for a selected cluster (in Figure 2, the bottom cluster is selected), information about it, including an address (achieved by reverse geocoding), and other properties of the cluster.

The user may also *pin* a cluster (see the button at the lower-right corner of the information panel of the bottom cluster). Upon pinning a cluster, the system presents all the clusters that are paired with the pinned cluster, to show all the places that are linked to the pinned cluster.

3.2 The Back-end

The back-end contains a server, a database, a module for crawling and storing the data and a module for computing clusters and links between them. The modules are implemented in Java.

Crawling and Storing Geo-Tagged Messages. Geo-tagged messages are retrieved from social networks using their public APIs. Our current dataset consists of messages from Twitter and Instagram. Each message is associated with several properties such as the location of the message, a caption or text, posting time, etc. Each message is associated with a user that has a unique user id. Hence, we are able to retrieve all the messages sent by a specific user.

The Database. Retrieved messages are stored in a MongoDB database system (`http://www.mongodb.org`). MongoDB is an open-source document-based database. It is suitable for storing a large quantity of messages and it provides an intuitive API and easy integration with Java. It conducts concurrency control (transaction management) in an effective way that allows to effectively retrieve data while processing analysis tasks or presenting results to the user.

There are two indexes to support efficient retrieval of messages according to different parameters. A spatial index, in the form of a quad-tree, supports efficient retrieval of messages from a specific area. A temporal index supports retrieval of messages that were sent in near times—the index resembles an array where each cell contains references to the messages posted during the time interval the cell represents.

The Server. At the back-end there is a Restlet server that provides a REST API for supporting the client requests (`http://restlet.org`). The main functions of the API include *(1)* specifying and initiating a new data analysis task, and *(2)* retrieving from the database and presenting the results of executed tasks. Since the computation of data analysis tasks may take time, upon a request for a new data analysis task, the server runs a new thread to handle the task (according to the parameters of the request).

When the computation ends, the revealed pairs are stored in the database. Each cluster entry contains an identifier, the center-of-mass of the cluster, the coordinates of the bounding box of the messages of the cluster (for efficient search), the number of messages in the cluster and the number of users associated with the cluster. Upon a request for a complete analysis task, the server retrieves the results from the database and sends them to the client for presentation.

4. DISCOVERING LINKED PLACES

We briefly present our methods of finding linked places.

4.1 Clustering Geo-Tagged Messages

In our task, the number of messages can be large, so we had to use an efficient clustering algorithm. Since in an analysis task the number of clusters is initially undefined, we used a version of *agglomerative clustering*, which is a type of hierarchical clustering [8]. Agglomerative clustering, in contrast to divisive clustering, is a "bottom up" process. Initially, each point is considered a cluster. Then, repeatedly, a cluster is chosen and merged with the cluster closest to it. (The new cluster replaces the two merged clusters, thus after each merge step, the number of clusters is decreased by one.) In this process, the main challenge is to guarantee a cluster represents a single place. To do so, we employ two stopping conditions—one that is based on a distance threshold and one that is based on *location-name preservation*. The distance threshold prevents merging two clusters for which the distance between their centers-of-mass exceeds the threshold. In the location-name preservation, initially we assign a place name to each cluster, by applying reverse geo-coding on the center-of-mass of the cluster. We avoid merging clusters if their location names do not match. The clustering process ends when no pair of clusters can be merged.

To effectively retrieve the messages from the database, the system only retrieves messages whose bounding box intersects the analyzed area (city). Another simplification, to improve the efficiency, is retrieving from the database only users whose number of messages exceeds a given threshold, to avoid dealing with users who visited very few places.

Since the clustering algorithm has $O(n^3)$ time complexity, where n is the number of messages in the dataset, we used thresholds and other simplifications to make the process feasible. In this process, an iteration is a merge of two clusters or marking a cluster as part of the output and removing it from the set of processed clusters. Initially, there are n clusters, and each iteration decreases by one the number of processed clusters—either by a merge or by moving a cluster to the output. So, there are at most n iterations. In each iteration, each cluster is only considered as a potential merge for a constant number of clusters in its vicinity, and a cluster that cannot be merged with any cluster is removed from the set of processed clusters. Thus, each iteration, among the n iterations, has a time complexity that does not depend on n, and the time complexity is $O(n)$.

4.2 Finding Links Between Clusters

In order to present results that can be examined by an urban planner, the final number of clusters should be limited. The goal is to present only *significant* clusters. In the definition of a task, the urban planner can specify the desired number of clusters in the result. We denoted this number by k. The question is how to choose k significant clusters from the clustered computed by the clustering algorithm.

The system employs two heuristics for choosing the significant k clusters. One approach is to return the k clusters with the highest number of messages. This approach is simple but it can be biased in a case where a few users posted many messages from the location of the cluster. Another approach is to count for each cluster the number of unique users who have a message in the cluster. Then, return the k clusters with the highest count of unique users.

For the k selected clusters, the system computes for each pair the number of mutual users. Consider two clusters—C_1 with c_1 users and C_2 with c_2 users—and m mutual users in C_1 and C_2. Then, the *ratio of mutual users* is defined as $\max\left\{\frac{m}{c_1}, \frac{m}{c_2}\right\}$, so if a large number of users who visited one place also visited the other place, then the two places should be considered as linked, even if the second place is a very large cluster and is linked to several places. To effectively present the results to the user, the pairs are sorted according to the ratio of mutual users.

To compute the number of mutual users, the system applies the following approach. First, an array A_c is created with an entry for each pair of clusters. Then, the system creates a bipartite graph B, where the nodes are all the clusters and all the users, and there is an edge between a cluster and a user if the user has a message in the cluster. After creating B, in a single pass over B the system finds all (C_1, C_2, u) such that there are edges (C_1, u) and (C_2, u) in B. For each such triple, the entry for C_1 and C_2 in A_c is increased by 1. At the end of this process, the k cluster pairs with the highest count in A_c are returned.

5. DEMONSTRATION

Our demonstration presents the City Nexus system. We show how a large number of messages can be collected, clustered, and analyzed using different parameters, for finding jointly-visited places in different cities in the world (New York, Los Angeles, London). The system illustrates an intuitive approach for presenting connected places on a map while allowing novice users to control various parameters.

A video of the system is available via the following link: `http://www.youtube.com/watch?v=nUbc4uqsprs`, illustrating a discovery of linked locations, based on Twitter tweets in New-York City. The clustering algorithm starts with 10000 users, filtering out users with less than 10 messages. Messages are clustered within a radius of 300 meters. Then, only the 30 clusters with the highest number of users are selected for the next step. From the optional pairs, we filter out pairs of clusters that the distance between them is less than 4 kilometers and clusters that do not have at least 15% of mutual users (this is based on arbitrary parameters and can be changed). The result pairs are presented, sorted by the percentage of mutual users. A few discovered pairs are obvious, e.g., many people who visited Times Square also visited SoHo, Manhattan, and such pairs provide a verification of the approach. Most of the pairs, however, are pairs we could not predict, and such pairs provide new insights about the city. The demonstration will present this search and additional searches with other parameters, over New York, London and Los Angeles.

6. ACKNOWLEDGMENTS

This research was supported in part by the Israel Science Foundation (Grant 1467/13) and by the Israeli Ministry of Science and Technology (Grant 3-9617).

7. REFERENCES

[1] A. Bawa-Cavia. Sensing the urban: using location-based social network data in urban analysis. In 1^{st} *Workshop on Pervasive Urban Applications*, 2011.

[2] J. Cranshaw, R. Schwartz, J. I. Hong, and N. M. Sadeh. The livehoods project: Utilizing social media to understand the dynamics of a city. In *International AAAI Conference on Weblogs and Social Media*, 2012.

[3] Y. Doytsher, B.Galon, and Y. Kanza. Storing Routes in Socio-spatial Networks and Supporting Social-based Route Recommendation. In *Proc. of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, pages 49–56, 2011.

[4] L. Ferrari, A. Rosi, M. Mamei, and F. Zambonelli. Extracting urban patterns from location-based social networks. In *Proc. of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, pages 9–16, 2011.

[5] B. Hu and M. Ester. Spatial topic modeling in online social media for location recommendation. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 25–32. ACM, 2013.

[6] F. Kling and A. Pozdnoukhov. When a city tells a story: Urban topic analysis. In *Proc. of the 20th ACM SIGSPATIAL International Conf. on Advances in Geographic Information Systems*, 2012.

[7] C. Ratti, R. M. Pulselli, S. Williams, and D. Frenchman. Mobile Landscapes: using location data from cell phones for urban analysis. *Environment and Planning B: Planning and Design*, 33(5), 2006.

[8] R. Xu, D. Wunsch, Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.