DBLOC: Density Based Clustering over LOCation Based Services

Yeshwanth D Gunasekaran †, Md Farhadur Rahman †, Sona Hasani †, Nan Zhang ‡, Gautam Das †
†The University of Texas at Arlington
‡The Pennsylvania State University
(yeshwanth.durairajgunasek,mdfarhadur.rahman,sona.hasani)@mavs.uta.edu,nan@ist.psu.edu,gdas@uta.edu

ABSTRACT

Location Based Services (LBS) have become extremely popular over the past decade. Popular LBS run the entire gamut from mapping services (such as Google Maps) to restaurants reviews (such as Yelp) and real-estate search (such as Zillow). The backend database of these applications can be a rich data source for geospatial and commercial information such as Point-Of-Interest (POI) locations, reviews, ratings, user geo-distributions, etc. However, access to the backend database is often restricted by a public query interface (often web-based) provided by the LBS owners. In most cases the public search interface of these applications can be abstractly modeled as *k*NN interface, taking a geolocation (i.e., latitude and longitude) as input and returning top-*k* POI's that are closest to the query point, where *k* is a small constant such as 50 or 100. Because of this restriction it becomes extremely difficult for third-party users to perform analytics or mining over LBS.

We demonstrate DBLOC, a web-based system that enables analytics over the LBS by using nothing but limited access to *k*NN interface provided by the LBS. Specifically, using DBLOC the users can perform density based clustering over the backend database of LBS. Due to *query rate limit* constraint - i.e., maximum number of *k*NN queries a user/IP address can issue over a specific period of time, it is often impossible to access all the tuples in backend database of an LBS. Thus, DBLOC aims to mine from the LBS a cluster assignment function $f(\cdot)$, such that for any tuple t in the database (which may or may not have been accessed), $f(\cdot)$ can produce the cluster assignment of t with high accuracy. We also demonstrate how DBLOC enables the users to further analyze the discovered clusters in order to mine interesting intra/inter cluster information.

ACM Reference Format:
Yeshwanth D Gunasekaran †, Md Farhadur Rahman †, Sona Hasani †, Nan Zhang ‡, Gautam Das †. 2018. DBLOC: Density Based Clustering over LOCation Based Services. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193561

1 INTRODUCTION

In this paper, we present DBLOC, a system that enables analytics over the backend database of real world location based services (LBS). Specifically, DBLOC allows the users to perform *spatial clustering* over the points of interests (POIs) stored in the backend database of LBS using nothing but limited access to the *k*NN interface provided by the LBS. In addition to clustering, the system allows analytics over the output clusters such as comparing the output clusters over a specific attribute, identifying interesting attributes of each clusters, etc. DBLOC now works with a suite of popular LBS systems and features such as Google Maps[1], Flickr[2], and popular real-estate site Zillow[3].

Location Based Services (LBS): LBS have become extremely popular in recent years. Popular LBS run the entire gamut from mapping services (such as Google Maps) to restaurants reviews (such as Yelp) and real-estate search (such as Zillow). The backend database of these LBS stores tuples that correspond to POIs (i.e., restaurants, houses, gas stations, etc.) or users of these applications. In addition to the location information (latitude and longitude), tuples in backend databases are often augmented with a set of *non-spatial* information. For example, for each restaurant type POI, Google Maps stores information such as restaurant name, rating, reviews, price level, and cuisine type, etc. However, access to the backend database of these applications is often restricted by a public query interface (often web-based) provided by the LBS. In most cases the search interfaces can be abstractly modeled as "nearest neighbor" *k*NN interface - taking a query point (i.e., latitude and longitude pair) as an input and returning k nearby POIs ordered according to their distance from query point, where k is a small constant such as 50 or 100.

Moreover, access to the backend database is further restricted by *constraints* such as *query rate limitation.* This limits the number of *k*NN queries a user or an IP address can issue over a specific period of time. For example, Google Maps API imposes a query rate limit of 10,000 per user per day. Since the *k*NN search interface of LBS applications returns only a small number of tuples (at most k) for each query, the query rate constraint makes crawling of backend database of LBS extremely difficult, if not impossible.

Motivation and Technical Novelty of DBLOC: The backend database of an LBS is often a gold mine of information for understanding the corresponding application domain. For example, data stored in real-estate LBS such as zillow.com offer critical insights into the geographic spread of wealth, education quality, etc. Similarly, city authorities or urban planners may find out interesting regions (i.e., regions with high user activity) in a city that require more attention by monitoring users of location based social networks. However, due to the aforementioned restrictions (top-k, query rate limit, etc.) imposed by the LBS applications, it becomes

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3193561

[1] https://www.google.com/maps
[2] https://www.flickr.com/
[3] https://www.zillow.com/

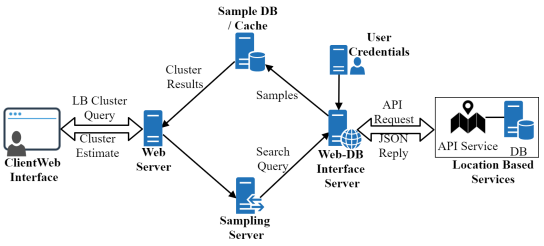

Figure 1: Architecture of DBLOC

extremely difficult for the third-parties to tap into the LBS for data analytic or mining purposes.

In contrast to existing demo systems [3] on spatial data, DBLOC enables the users to perform analytics over LBS data using nothing but the restrictive, public-access, *k*NN search interface provided by the LBS. Specifically, DBLOC enables third party users to perform spatial clustering over online LBS databases by issuing only a small number of *k*NN queries. Moreover, using DBLOC, the users can further analyze the discovered clusters in order to mine interesting intra/inter cluster information. For example, by analyzing the clusters over real-estate data such as Zillow, we can find the areas where citizens of different socioeconomic status live. Similarly, clustering over restaurants in a city (using Google Maps) may reveal different cuisine types that are popular in different areas of the city. The closest work to our demo system is ANALOC [5]. However, instead of clustering, ANALOC answers aggregate queries such as COUNT, SUM and AVG using the public *k*NN interface of LBS.

While many spatial clustering algorithms have been proposed in the literature, DBLOC uses the core concept of a popular density-based clustering algorithm DBSCAN [1]. Note that, objective of DBLOC is not to select best-performing algorithm for LBS data, but to demonstrate the feasibility of performing clustering using nothing but a few *k*NN query answers. The two most critical challenges faced by DBLOC for enabling spatial clustering over LBS are the *input* and *output* of the clustering algorithm. On the input side, there is no direct way for an LBS client to run the density-calculating queries required by DBSCAN. Similarly, on the output side, a faithful implementation of any spatial clustering algorithm requires accessing each tuple in database at least once. Thus, requiring the system to issue an inordinate amount of *k*NN queries, i.e., at least n/k, where n is the total number of tuples in database. Using the algorithms proposed in [4], DBLOC overcomes both of the aforementioned challenges and enables clustering using LBS.

2 SYSTEM ARCHITECTURE

Figure1 demonstrates the architecture of DBLOC. The system consists of five components: web server, sampling server, sample database, user credential database, web database, and interface server. Design and functionality of each component is described in the following subsections.

2.1 Web Server

The task of web server is to provide the users with a web interface that allows specification of clustering query (as input) and displays the cluster discovery process in real-time using Google Maps. Moreover, the web server provides additional controls in the input interface in order to support analytics over the discovered clusters.

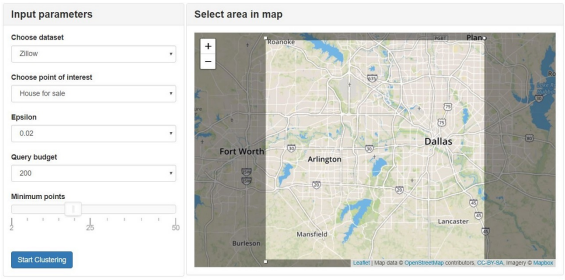

Figure 2: Input interface of DBLOC

Input Interface: DBLOC allows the users to specify the clustering query through an intuitive and step-by-step web interface. In order to start the clustering process, the users can specify (1) *data source*, i.e., choice of LBS, (2) *point of interest*, i.e., POI type over which the clustering should be performed, (3) *bounding box*, i.e., the area of interest, (4) *query budget*, i.e., maximum number of LBS queries that the system can issue during cluster discovery process. (5) *cluster parameters*, i.e., ϵ and $minPts$, the two input parameters required by underlying clustering algorithm HDBSCAN [4]. The system also provides an option to the users to automatically obtain the cluster parameters by sampling tuples from the selected LBS. The algorithm for parameter determination process is presented in 2.2. An example of clustering query over the houses in Dallas-Forth Worth metroplex area with $\epsilon = 0.02$, $minPts = 10$ and query budget = 300 is presented in Figure 2.

Output Interface: The output interface displays the clusters discovered by HDBSCAN algorithm using Google Maps. In addition, the system provides controls to the users to perform analytics over the discovered clusters. Using these controls a user can discover interesting information such as top-k clusters with highest aggregate value on a specific attribute, clusters that are different from their neighbors over a set of aggregate measures, etc. As HDBSCAN progresses, the map section visualizes the cluster discovery process in real time. Figure 3 displays the the clusters discovered by HDBSCAN. Among all the clusters, the top-k (k = 5) clusters with highest average house price are shown in the cluster statistics section of output interface (Figure 4). The users can set the value of k, choice of attribute and aggregate measure using the controls in "Cluster ranking parameters" panel. The "Cluster comparison" panel (Figure 5) allows the users to select any subset of the discovered clusters and compare them over the set of available attributes (attributes returned by LBS in the *k*NN query answer). For each axis, the aggregate values are normalized in range [0, 1]. For Google Maps data source DBLOC utilizes the *user reviews* associated with POIs in clusters in order to mine frequent keywords in reviews. This enables the user to identify prominent features of each cluster.

2.2 Sampling Server

The sampling server is the main component of DBLOC. The key task of sampling server is to execute the clustering query issued by the user, and to perform analytics over the discovered clusters.

2.2.1 Construct Cluster Assignment Function. Due to the aforementioned constraints (i.e., top-k, query rate limit, etc.) imposed by the LBS, it is often impossible to perform faithful execution of traditional clustering algorithms. This is because existing spatial

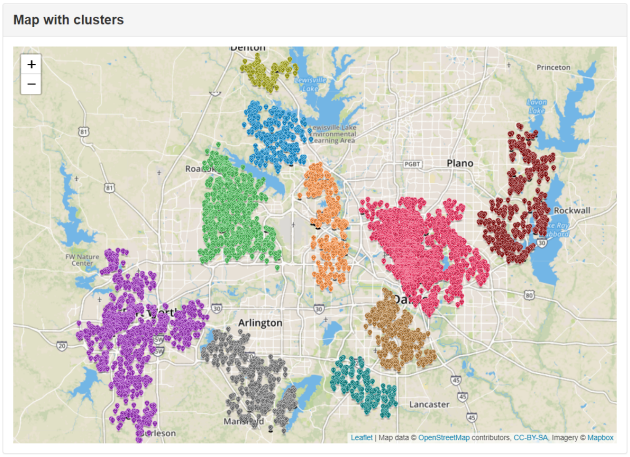

Figure 3: Clusters discovered by DBLOC

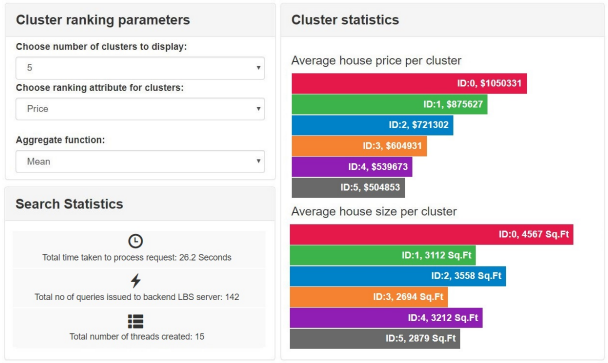

Figure 4: Cluster statistics

clustering algorithms need to access each tuple in database at least once, thus require issuing inordinate amount of LBS queries. The problem is resolved by adjusting both the *input* and the *output* of the clustering algorithm. Specifically, we propose HDBSCAN algorithm that instead of accessing all the tuples in a cluster, tries to discover the boundaries of the cluster by "skipping over" intermediate tuples. The details of the algorithm can be found in [4]. Here we only include a brief sketch of the techniques.

HDBSCAN: The baseline approach to simulate DBSCAN is to partition the 2D space into grids of length ϵ on each dimension and then generate the final clusters by merging dense grid cells that are adjacent to each other. The density property of a grid cell can easily be computed by issuing kNN query at the center of the grid cell. If out of the k returned tuples *minPts* of them fall inside the grid cell (assuming $k > minPts$) we can identify the cell as dense. However, this approach can be extremely inefficient since the total number of kNN queries need to be executed is equal to the number of cells in the grid. Hence, instead of identifying all of the dense cells in a cluster, we aim to find the *boundaries* of the cluster by "skipping over" adjacent cells. These boundaries serve as the clustering function $f(\cdot)$, since given a new tuple we can identify its corresponding cluster by checking whether it lies inside or outside of a cluster boundary. We start by designing boundary discovery algorithm for one dimensional data and then extend it to two dimensional space. In 1D, each cluster is essentially a segment of dense cells.

Figure 5: Cluster Comparison

The algorithm starts from a dense cell and identifies its left and right boundaries using the Boundary-Pursuit and C-Cell-Density-Test subroutines (details in [4]). While the boundaries of a cluster can be discovered by simply traversing left and right directions in 1D space, the directions that we need to follow for discovering a 2D cluster boundary is not very clear. This is mainly due to the different shapes a cluster can have in 2D space. The problem was solved by an innovative approach of mapping the points in 2D space to 1D using a *space filling curve* (SFC), and then discovering the clusters by running 1D clustering algorithm on the transformed space. Note that, the transformation of 2D space into 1D space may partition a 2D cluster into many 1D "min clusters". This is handled by a post-processing step of merging 1D clusters that are "close" to each other, eventually into 2D clusters. We also propose several optimization techniques in order to improve the performance of HDBSCAN algorithm. Please refer to [4] for further details.

Determining ϵ and *minPts*: The functionalities of the input parameters of HDBSCAN (ϵ and *minPts*) are similar to the parameters of DBSCAN - DBSCAN marks a point as dense if it contains more than *minPts* points inside its ϵ-neighborhood [1]. Therefore we use a similar heuristic approach proposed by DBSCAN to determine ϵ and *minPts* parameters for HDBSCAN. Specifically, we start by setting $minPts \leq k$ and then collect random samples from LBS using the kNN interface. Please refer to [2] for details about generating random samples form LBS. For each sample, we issue an LBS query with query point set to sample location and determine the distance between the sample and its *minPts*-th nearest neighbor. Finally the samples are sorted according their *minPts*-dist value and we choose the first point in the first valley of *minPts*-dist graph as ϵ value.

2.3 Web-DB Interface Server and User Credentials Database

The task of the web-DB interface server is two-fold: (1) to translate each kNN query issued by the sampling server to an LBS query, and (2) to parse the returned results, transform them to structured tuples, and pass them to the sample database component. Each LBS may require a different wrapper design for the kNN input/output translations. For example, a kNN query may be translated to simple HTTP GET or POST requests or RESTful API calls. The raw returned result from LBS is mostly in a structured format such as XML or JSON, which enables simple translation to our sample database.

Some LBS, however, return raw HTML code that has to go through DOM parsing and/or regular expressions before the structured tuples can be extracted.

The user credentials database component allows us to handle the query rate limit enforced by the LBS. Most LBS require logging in with user credentials for API and web access. In order to support multiple concurrent users, DBLOC uses user credential database to store the credentials of end users (such as API keys or username/password) and then issue queries on their behalf.

3 DEMO PLAN

In this section, we describe different scenarios that the users can use to interact with DBLOC. First, we shall describe the hardware setup, the backup plan in case of slow/non-existent Internet connections, and audience interaction. We then provide specific case studies for two real-world LBS Google Maps and Flickr.

3.1 Overview

Hardware Setup and Backup Plan: DBLOC is web-based and supports access from multiple platforms. During the demonstration, we shall provide laptops with access to DBLOC for visitors to interact with the system. The laptops are connected to the web server of DBLOC, which we shall host on Amazon EC2. As a backup during the demonstration session and in case of a disturbed or slow Internet connection, we intend to also host DBLOC locally on a separate demo laptop to serve as a local server simulating real-world LBS (using the real historical data we collected).

System Setup and Audience Interactions: While the design of DBLOC is generic to any LBS featuring a kNN search interface, the web-DB interface server component does require a pre-configured specification file for each LBS. The demo system shall contain a set of pre-configured specification files, including popular LBS ranging from online photo sharing services (such as Flickr) to map services (such as Google Maps and Bing) and real-estate search (such as Zillow). Visitors of the demo can select the LBS of interest, specify the clustering query, and interact with the system to analyze the discovered clusters.

3.2 Demonstration scenarios

Clustering POIs on Google Maps: Given a query point and an optional selection condition such as POI type - e.g., restaurant, gas station, etc., the Google Maps API returns at most $k = 60$ nearby POIs (matching the selection condition) ordered according to their distance from the query point. In addition, it also returns other relevant information about POIs such as user reviews, ratings, hours of operation, etc. We shall demonstrate the performance of DBLOC by clustering POIs on Google Maps. Sample clustering queries include clustering of restaurants in DFW area, clustering of gas stations in New York city, etc. Visitors will be able to visualize cluster discovery process, distribution of clusters over the query region, and statistics of each cluster. Moreover, the visitors can also start with clustering queries where the system will automatically determine the parameter values (ϵ and $minPts$) and then the users can play with the parameter values to get the desired result.

Clustering Photos in Flickr: In order to demonstrate the effectiveness of DBLOC to cluster any collection of geo-tagged items, we have included Flickr as one of the data sources of DBLOC. Flickr is a popular photo sharing service that provides API for getting photos taken near the query location. Users can issue at-most 3600 queries per hour using their API key. Each photo is also augmented with a set of tags that the users have used while sharing it. During the demonstration, the visitors can discover interesting places of a city by clustering photos in Flickr that are taken in that city. Moreover, we shall also showcase the popular tags used by the Flickr users for the photos inside each cluster.

Cluster Description: Knowing the popular words that are associated with a cluster can help the users to get a good picture of the overall characteristics of that cluster. For example, a cluster of restaurants that contains many Mexican cuisines may have words such as "tacos", "salsa", and "quesadillas" frequently present in the user reviews. In addition to answering the clustering query over Google Maps POI, we shall also showcase the frequent words associated with each cluster. This will help visitors to identify important characteristic of each cluster.

4 SUMMARY

We propose to demonstrate DBLOC, a system for performing clustering over Location Based Services. Moreover, the system also enables analytics over the discovered clusters, thus helping the users to get more insight about the output. In contrast to previous systems, DBLOC does not assume full access to the underlying data and requires noting but limited access to kNN interface provided by the LBS.

5 ACKNOWLEDGMENTS

The work of Yeshwanth D Gunasekaran, Md Farhadur Rahman, and Gautam Das was supported in part by the National Science Foundation under grant 1745925, the Army Research Office under grant W911NF-15-1-0020, a grant from Microsoft Research, and a grant from AT&T. This contribution was made possible in part by NPRP grant #07-794-1-145 from the Qatar National Research Fund (a member of Qatar Foundation). Sona Hasani was supported in part by NSF 1745925, a grant from Microsoft Research, and a grant from AT&T. Nan Zhang was supported in part by NSF under 1343976, 1443858, 1760059, ARO under W911NF-15-1-0020. The statements made herein are solely the responsibility of the authors.

REFERENCES

[1] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[2] W. Liu, M. F. Rahman, S. Thirumuruganathan, N. Zhang, and G. Das. Aggregate estimations over location based services. *Proceedings of the VLDB Endowment*, 8(12):1334–1345, 2015.

[3] A. Magdy, L. Alarabi, S. Al-Harthi, M. Musleh, T. M. Ghanem, S. Ghani, S. Basalamah, and M. F. Mokbel. Demonstration of taghreed: A system for querying, analyzing, and visualizing geotagged microblogs. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1416–1419. IEEE, 2015.

[4] M. F. Rahman, W. Liu, S. B. Suhaim, S. Thirumuruganathan, N. Zhang, and G. Das. Density based clustering over location based services. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 461–472. IEEE, 2017.

[5] M. F. Rahman, S. B. Suhaim, W. Liu, S. Thirumuruganathan, N. Zhang, and G. Das. Analoc: Efficient analytics over location based services. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1366–1369. IEEE, 2016.