ENABLING THIRD PARTY SERVICES OVER DEEP WEB DATABASES AND LOCATION BASED SERVICES

by

YESHWANTH DURAIRAJ GUNASEKARAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2018

Acknowledgements

It’s a great pleasure to acknowledge my deepest thanks and gratitude to my Advisor, Dr. Gautam Das, for giving me the opportunity to work with him and also for his constant support and guidance throughout my Master's thesis. I feel extremely thankful to be a part of DBXLAB, where I have had the opportunity to explore new horizons, which I never knew existed. I am also extremely grateful to Dr. Chengkai Li and Dr. Jia Rao for accepting to be on the thesis committee.

I sincerely appreciate the efforts of Dr. Abolfazl Asudeh, Mr. Farhadur Rahman and Ms. Sona Hasani of DBXLAB for their remarkable guidance and support throughout this project. It was wonderful working with all the members of the lab throughout my time as a DBXLAB member and I am extremely grateful for that.

Finally, I would like to thank my Parents and friends for the moral support and patience they have showered me with. You have all helped me focus in what has been a hugely enriching and rewarding process.

March 26, 2018

Abstract

ENABLING THIRD PARTY SERVICES OVER DEEP WEB DATABASES AND LOCATION BASED SERVICES

Yeshwanth Durairaj Gunasekaran, MS

The University of Texas at Arlington, 2018

Supervising Professor: Gautam Das

Deep web databases are pillars of today's internet services hidden behind HTML forms and Top-K search interfaces. While Top-K search interfaces provide a good way to retrieve information, it still lacks in addressing the diverse preferences of the users. Due to query rate limit constraint - i.e., maximum number of k-Nearest Neighbors queries a user/IP address can issue over a specific period of time, it is often impossible to access all the tuples in backed database. With the query rate limit constraint in mind, our motivation is twofold (i) Enable users to obtain individual records from these databases and rank them according to the user's preference, (ii) Enable the user to access aggregate information over these databases.

We introduce QR2 and DBLoc, both these systems access the hidden databases via their public search interfaces and operate without any knowledge on the underlying system ranking function. While QR2 helps in ranked retrieval of single tuples, DBLoc helps in aggregating information over Location based services.

QR2 enables on-the-fly processing of queries with any user-specified ranking functions (with or without selection conditions), no matter if the ranking function is supported by the database or not. Using DBLoc the users can perform density based

clustering over the backend database of Location Based Services. Thus, DBLOC aims to mine from the LBS a cluster assignment function f (·). We have developed an efficient system for both these problems to be scalable, reliable and secure. We also support multi user accessibility for both these systems and illustrate how to efficiently deploy them in the industry.

Table of Contents

List of Illustrations

Chapter 1

INTRODUCTION

1.1 Motivation

Deep web databases are pillars of today's internet services hidden behind HTML forms and Top-K search interfaces. Content of the deep web can be located and accessed by a direct URL or IP address, and may require passwords or other security access past the public website page. One of the major limitations of Deep web databases lies in retrieving the information from these databases. Since these databases have a limited query interface i.e. limited by either applying a query rate limit per IP or query rate limit per user, we cannot easily get the information that we are looking for. There is also no way of getting aggregate information from these hidden databases. With that said, let us dive into looking at two different approaches at getting the required data from deep web databases.

1.1.1 Retrieval of single tuples

The ranked retrieval model has rapidly replaced the traditional Boolean retrieval model as the de facto way for query processing in client-server (e.g., web) databases. Unlike the Boolean retrieval model which returns all tuples matching the search query selection condition, the ranked retrieval model orders the matching tuples according to an often proprietary ranking function, and returns the top-k tuples matching the selection condition (with possible page-turn support for retrieving additional tuples). The ranked retrieval model naturally fits the usage patterns of client-server databases. For example, the short attention span of clients such as web users demands the most desirable tuples to be returned first. In addition, to achieve a short response time (e.g., for web databases), it is essential to limit the length of returned results to a small value such as k. Nonetheless, the ranked retrieval model also places more responsibilities on the web database designer, as the ranking function design now becomes a critical feature that must properly capture the need of database users. In a practical situation, different users often have diverse and

sometimes contradicting preferences on numerous factors. Even more importantly, many database owners simply lack the expertise, resources, or even motivation (e.g., in the case of government web databases created for policy or legal compliance purposes) to properly study the requirements of their users and design the most effective ranking functions. For example, many flight-search websites, including Kayak, Google Flights, Sky Scanner, Expedia, and Priceline offer limited ranking options on a subset of the attributes, that, for example, does not help ranking based on cost per mileage. Similar limitations apply to the websites such as Yahoo! Autos (resp. Blue Nile), for ranking, for example, based on mileage per year (resp. summation of depth and table percent). As a result, there is often a significant gap, in terms of both design and diversity, between the ranking function(s) supported by the client-server database and the true preferences of the database users. Thus we propose a system **QR2** [1] which has the objective to rerank query results obtained from these hidden web databases. Our system will take in a bunch of filtering and ranking criteria and output the results based on the ranking function. We support both one dimensional and multi-dimensional ranking. E.g. One dimensional- (Sort based on price)

E.g. Multi-dimensional ranking (Sort based on function Price + house Size) for Zillow. We have developed algorithms [2] for reranking and also systems to correctly get the user input and efficiently rerank results in a scalable and reliable manner.

1.1.2 Retrieve aggregate information

In this section we will talk about retrieval of aggregate information, for this case we will look at the specific problem of LBS. LBS systems have a plethora of information, namely where the backend database contains Point-of-Interest locations which has various other attributes attached to t_i other than the location itself. Moreover, we also need to understand is that the retrieval model of the LBS systems do not resemble that of a traditional database which returns tuples based on a single ranking function. Instead, LBS systems return POIs in a kNN fashion. More specifically,

our objective is to study a novel problem of enabling spatial clustering over an online LBS database by issuing only a small number of kNN queries supported by the LBS interface. Clustering is one of the key problems in spatial data mining, with a wide range of applications. For example, by performing clustering over the geocoded tweets at Twitter, a third party may identify hotspots or popular events. Similarly, clustering over real-estate data such as Redfin can unveil the areas where citizens of different socioeconomic status live. While many spatial clustering algorithms have been studied in the literature, the objective is not to select the best-performing algorithm for LBS data, but to instead demonstrate the feasibility of enabling spatial clustering using nothing but a few kNN query answers. For this purpose, we consider as a baseline a fundamental yet popular density-based clustering algorithm, DBSCAN [3], and develop a DBSCAN-like algorithm for LBS data with only a kNN interface for data access. While we focus on developing a DBSCAN-like algorithm, we propose a system **DBLOC** where we have also addressed the challenges in building such a system in a scalable and reliable manner. We have included other features such as how to make sense out of these arbitrary clusters.

Chapter 2

TECHNICAL BACKGROUND

2.1 QR2

Before we dive into the algorithms or the implementation let us take a look at some of the required technical background necessary.

2.1.1 Database model

Let us look at how to define the client server database. We will now formally define the how to represent the attributes, domains and query interface. Consider a client-server database D with n tuples over m ordinal attributes $A_1 \ldots A_m$. Let the value domain of A_i be V (A_i) = {v_{i1}; : : : ; v_{ij}}. The database may also have other categorical attributes $B_1 \ldots B_m$. But since they are usually not part of any ranking function, they are not the focus of our attention for the purpose of this paper. We assume each tuple t to have a none-NULL value on each (ordinal) attribute Ai, which we refer to as t[Ai]. Note that if NULL values do exist in the database, the ranking function usually substitutes it with another default value (e.g., the mean or extreme value of an attribute). In that case, we simply consider the occurrence of NULL as the substituted value. We make the assumption that each tuple has a unique value on each attribute, before introducing a simple post-processing step that removes this assumption in

2.1.2 Query Interface

Most client-server databases allow users to issue certain “simplistic” search queries. Often these queries are limited to conjunctive ones with predicates on one or a few attributes. Examples here include web databases, which usually allow such conjunctive queries to be specified through a form-like web search interface. Formally, we consider search queries of the form shown in Figure 1.

q: SELECT * FROM D WHERE $A_{i_1} \in (v_{i_1}, v'_{i_1})$ AND $\cdots$ AND $A_{i_p} \in (v_{i_p}, v'_{i_p})$ AND *conjunctive predicates on* $B_1, \ldots, B_{m'}$, where $\{A_{i_1}, \ldots, A_{i_p}\} \subseteq \{A_1, \ldots, A_m\}$ is a subset of ordinal at-tributes, and $(v_{i_j}, v'_{i_j}) \subseteq V(A_{i_j})$ is a range within the value domain of A_{i_j}.

Figure 1 Query representation

2.2 DBLoc

We will continue to explore both QR2 and DBLOC in parallel, while we have taken a look at the required technical background for QR2, let us get to know about LBS systems, KNN query interface, DBSCAN and Space Filling curves.

2.2.1 Location Based services

Real-world LBS provide search and recommendation for numerous types of geospatial and commercial information such as Points-of-Interest (POIs), restaurants, real-estate properties, etc. Popular examples range from mapping services (e.g., Google Maps) to restaurants reviews (e.g., Yelp) to real-estate search (e.g., Redfin). Besides these dedicated LBS systems, LBS-related features have been widely integrated into other web based systems, e.g., social media platforms such as Twitter, WeChat, Sina Weibo, etc. Generally speaking, each LBS has a backend database where each tuple represents a geotagged entity (e.g., a POI in mapping services or a user in social media). Attributes of a tuple often capture both geographical coordinates (e.g., latitude and longitude) as well as other structured information such as POI name, review ratings, etc. Public access to an LBS database is usually limited to a web (or API) based search interface. Such an interface often allows only k-Nearest-Neighbor (kNN) queries - i.e., upon given a geolocation p and, optionally, a selection condition s, the interface returns a small number (up to a pre-determined constant k such as 20 or 50) of tuples in the database that, among those matching the selection condition s, are closest geographically) to p.

2.2.2 DBSCAN

Density-based spatial clustering of applications with noise or DBSCAN is a density based clustering algorithm. it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN requires two parameters: ε (eps) and the minimum number of points required to form dense region (minPts). It starts with an arbitrary starting point that has not been visited. This point's ε-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. [3] Note that this point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

Let us assume a case where a point is found to be in a dense cluster. The ε-neighborhood is also part of that cluster. Therefore, every point that is found within the ε-neighborhood is added, as is their own ε-neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise. DBSCAN can be used with any distance function (as well as similarity functions or other predicates). The distance function (dist) can therefore be seen as an additional parameter.

2.2.3 Space Filling Curves

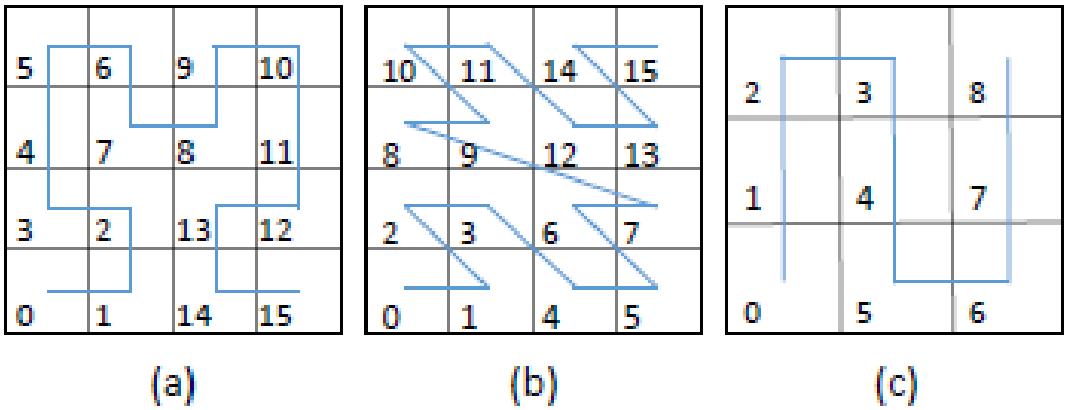


Figure 2 Figure shows space filling curves (1) Hilbert (b) Z-Curve (c) Peano

We recommend the reader is also comfortable with space filling curves. A space filling curve is a curve in 2D or 3D space which is continuous. Take a look at Figure 2 (a), (b) or (c) a continuous curve fills the entire 3x3 grid. Since this curve is continuous we can number the grids starting from 0 to N. One of the interesting properties of the space filling curve is that once the grid is numbered, it can be pulled or transformed into a 1D grids of size 0 to N^2.

Other interesting properties of Space filling curves include when two cells are close to each other in One dimension they are also close to each other on 2D grid space. But the inverse of this statement (Grids close in 2D are close to each other in 1D) cannot be said to be true. A well designed SFC guarantees that two points close to each other in the mapped 1D space are also close together in the original 2D space. This property fits our purpose since we can skip points (by binary search) that are close to each other in mapped 1D space as they will also be potentially inside the same cluster in original 2D space. (we will cover this later when we look at methodologies) However, we must caution that points that are close to each other in 2D pace might not be close in the mapped 1D space. [4] [5]

A curve (with endpoints) is a continuous function whose domain is the unit interval [0, 1].

In the most general form, the range of such a function may lie in an arbitrary topological space, but in the most commonly studied cases, the range will lie in a Euclidean space such as the 2-dimensional plane (a planar curve) or the 3-dimensional space (space curve).

Sometimes, the curve is identified with the range or image of the function (the set of all possible values of the function), instead of the function itself. It is also possible to define curves without endpoints to be a continuous function on the real line (or on the open unit interval (0, 1)). [6]

Chapter 3

METHODOLOGIES

3.1 QR2

3.1.1 One Dimensional Reranking

In this section we will try to come up with new algorithms for 1D type problems. Let us define 1D type problems first, here the basic input is going to be of type

1. Filtering Conditions
2. Ranking Function – This will be of the form E.g. for Bluenile (Desc Carat)

1D Baseline

- Step1: Need to find t_h, issue query q_0.
- Step2: Keep narrowing the search space until t_h is found, use the system provided predicates.
- Limitations: The query cost depends on the correlation between the system ranking function and the attribute.

Figure 3 1D Baseline

1D Binary

- Do a binary search on the space, issue query q_0.
- Keep narrowing the search space until t_h is found.
- Use the system provided predicates.
- Limitations: Query cost is high when
    - System function is negatively correlated with the attribute (A_i)
    - There are densely clustered tuples with extremely close values on A_i.

Figure 4 1D Binary

1D Rerank

Since the main limitation of 1D binary is when we meet densely clustered tuples extremely close. Let us assume we have an Oracle which exists, which will return the appropriate values when we meet a dense region. Now let us focus on designing the Oracle itself.

Oracle Design: If the oracle does not exist for a region, call 1D-BASELINE to index it on the y. Applying the proper parameter settings, 1D-RERANK is in O(log(n)). The Get-next operation designed by 1D-RERANK can get used to develop a sorted-access top-k algorithm (e.g. TA) on top of it and enable HD Get-next. This approach has a major efficiency problem, mainly because of not leveraging the full power provided by client-server databases.

3.1.2 Multi-Dimensional Ranking

In this section we will try to come up with new algorithms for MD type problems. Here the basic type of input if going to be

1. Filtering Conditions
2. Ranking Function – This will be of the form E.g. for Bluenile (Price - 0.6 Carat)

Here the significance of a negative in the function means that we score a tuple based on the least price and best carat. Meaning such a function would return the diamonds with least price and biggest carat. But for the sake of simplicity let us look at a scenario where both attributes A_i and A_j have both positive coefficients.

MD Binary

Let us understand the notion of direct domination detection: When a query such as q_2 returns a tuple t_0 that ranks lower than t, whether this is by the absence of higher-ranked tuples in q_2, or by the ill conditioned nature of the system ranking function. Virtual Tuple Pruning: Prune the search space according a virtual tuple created for the purpose of minimizing the pruned subspace.

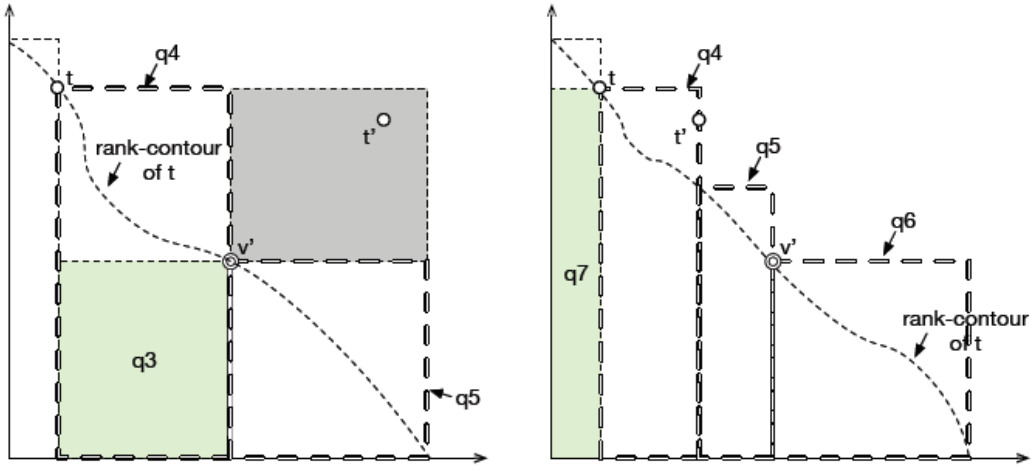


Figure 5 MD Binary

Once the rank contour is established, we find a virtual tuple v`. v` is found such that it covers the maximum area under the rank contour. Thus allowing us to eliminate maximum search space before issuing other queries (sort of like a binary search).

MD Rerank

This approach is very similar to the 1D re rank approach where we have an oracle design, we use MD binary to identify the tuples and when we touch a dense area we store them and crawl them using MD baseline.

On-The-Fly Indexing: proactively record as an index densely located tuples once we encounter them, so that we do not need to incur a high query cost every time a query q triggers visits to the same dense region. High-level idea: follows MD-BINARY until a remaining search space

(1) is covered by a region in the index

(2) has volume smaller than the threshold.

3.2 DBLoc

3.2.1 HDBScan 1D

- Randomly Select a cell q and check its density property.
- If q contains more than $minPts$ POIs inside, discover the boundaries of cluster containing q.
    - Repeat until left (resp. right) boundary is found.
    - a = Right (resp. left) most sparse cell on the left (resp. right) side of q.
    - s = Binary search on range $(a, q]$ (resp. $[q, a)$)
    - Validate the continuity of the range by C-cell density test.

- Given a new tuple t, assign it to the appropriate cluster using the cluster boundaries discovered.

Figure 6 One Dimensional clustering

3.2.2 HDBScan 2D

- Partition the grid space into ε x ε space.
- Apply SFCs such as Hilbert curve or Z- curve.
- Map 2D to 1D Space and perform clustering.
- SFCs ensure that cells close by in 1D space are always close by in 2D space too.

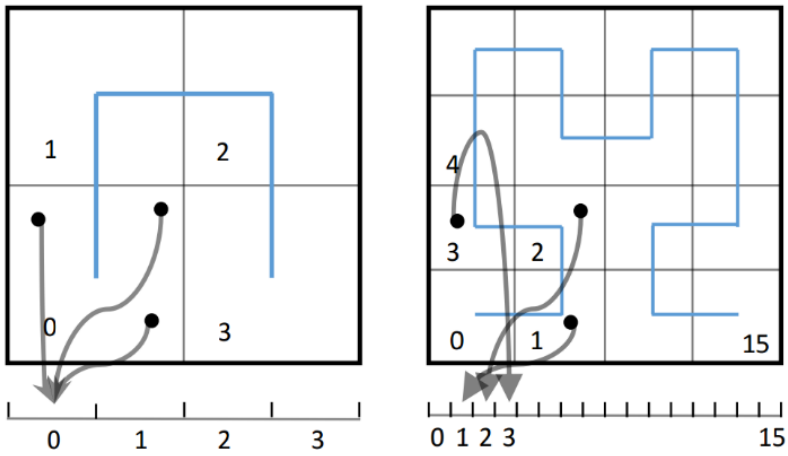


Figure 7 Mapping 2D to 1D

3.2.3 Adaptive SFC

- Problem: If ε is small, large number of grid cells defined by the SFCs. And increases query cost.
- What we need: Finer εxε cells in dense region and large cells in otherwise sparse regions.
- Adaptive SFC: shape of the adaptive SFC depends on the underlying data distribution

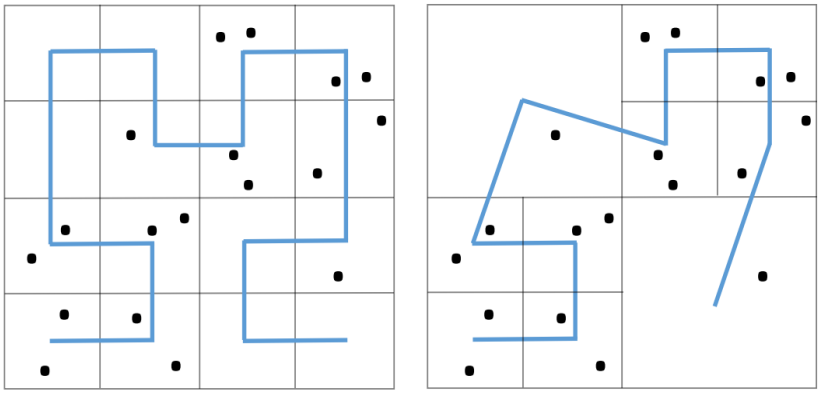


Figure 8 Adaptive SFC

Chapter 4

TECHNICAL CHALLENGES AND FEATURES

4.1 QR2

4.1.1 Parallel Processing

QR2, as a third party service, may have different users issuing different queries at the same time; therefore, the sequential processing of queries may significantly reduce the system performance. In addition to a non-sequential processing of different queries, we apply parallel processing while performing each query, in order to reduce the query processing time. We note that this may, sometimes, increase the number of queries issued to the web database. Specially, the following parallel processing's help reducing the effect of the web database delay:

– In order to verify that the top discovered tuple is indeed the true top one, we issue several queries, in parallel, that cover the areas in which a tuple may dominate the discovered tuple.

– In MD, after the initial get-next, in order to discover subsequent tuple(s), the algorithm partitions the search space on an attribute A_i and searches the two subspaces independently. The subsequent tuple is the top tuple from these two regions with the best score. Since the search in subspaces is done independently, it is easily parallelable.

4.1.2 Cardinality Issue

Handling the attributes with different domains is left as a part of ranking function design in [2]. However, in practice, it does not seem realistic to expect the users to take the burden. Thus, we apply the min-max normalization of attributes values to resolve this issue. Please note that obtaining the min and max values on each attribute is simply doable using the 1D-RERANK algorithm.

4.1.3 General positioning assumption

A general assumption in [2] is that no two tuples have the same values on a given attribute. This assumption, however, may not hold in practice. Especially when the number of tuples matching the predicate $t[A_i] = Vc$ is greater than system-k, the issued query to the web database never underflows. To solve this, we implemented the crawling algorithm proposed in [7]. QR2 calls this function when the number of tuples matching a value Vc is greater than system-k.

4.2 DBLoc

4.2.1 Merging 1D Clusters

We now consider how to merge the mini-clusters generated by HDBSCAN-1D to real 2D clusters. A simple approach here is to merge mini-clusters containing grids that neighbor each other in 2D (i.e. grids that share an edge). A problem with this solution, as we found through experiments, is that it is likely to merge two clusters into one if the clusters happen to be close to each other. Setting the _ value very small might solve this, but this will also increase the query cost. To overcome this problem, we select a subset of points from each clusters as representative to compute the inter cluster distance. This is similar in nature to the concept of using fixed set of representative points to measure cluster distance in CURE a hierarchical clustering algorithm. Specifically, we compute the l-distance between two mini-clusters - i.e., we identify all points in the two mini-clusters that have been observed in previous query answers. Then, we select the top-l pairs of points with the minimum distance from each other, and compute their average distance. We merge the two mini-clusters if their l-distance falls below a pre-determined threshold. Setting a small value of l

would split larger clusters as they do not capture the shape of the clusters. Empirically, we found that setting l to minPts/2 provided best results.

4.2.2 Auto detect Epsilon and Min pts

In this section, we develop a simple but effective heuristic to determine the parameters Eps and MinPts of the "thinnest" cluster in the database. [3] This heuristic is based on the following observation. Let d be the distance of a point p to its k-th nearest neighbor, then the d-neighborhood of p contains exactly k+l points for almost all points p. The d-neighborhood of p contains more than k+l points only if several points have exactly the same distance d from p which is quite unlikely. Furthermore, changing k for a point in a cluster does not result in large changes ofd. This only happens if the k-th nearest neighbors ofp for k= 1,2, 3 .... are located approximately on a straight line which is in general not true for a point in a cluster. For a given k we define a function k-dist from the database D to the real numbers, mapping each point to the distance from its k-th nearest neighbor. When sorting the points of the database in descending order of their k-dist values, the graph of this function gives some hints concerning the density distribution in the database. We call this graph the sorted k-dist graph. If we choose an arbitrary point p, set the parameter Eps to k-dist(p) and set the parameter MinPts to k, all points with an equal or smaller k-dist value will be core points. If we could find a threshold point with the maximal k-dist value in the "thinnest" cluster of D we would have the desired parameter values. The threshold point is the first point in the first "valley" of the sorted k-dist graph. All points with a higher k-dist value (left of the threshold) are considered to be noise, all other points (right of the threshold) are assigned to some cluster.

Chapter 5

ARCHITECTURE

5.1 QR2 Architecture

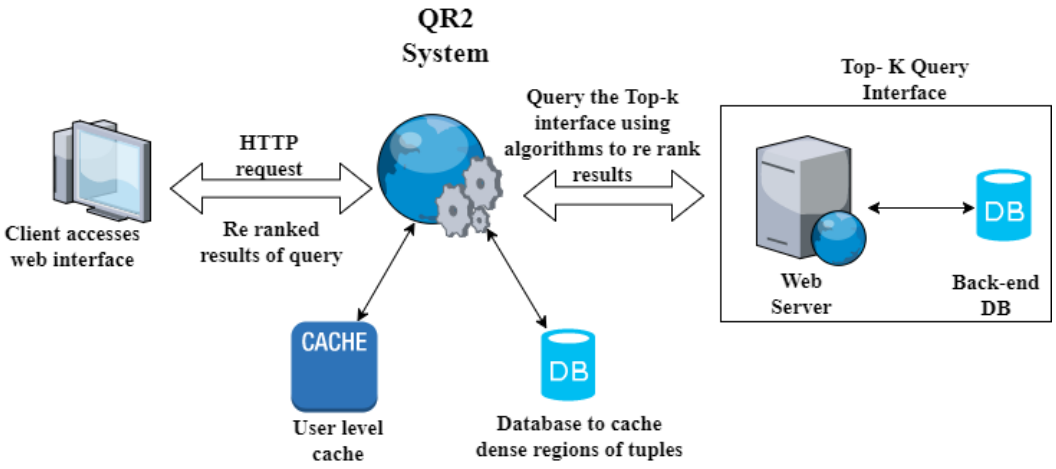


Figure 9 Architecture of QR2 system

Figure 9 shows the architecture of QR2. Web service is the central component of the architecture, where the users connect via Internet and select the data source. Once a user submits a query along with a ranking preference, the server creates a new session and processes the users request. The session variable (user level cache) is used to store the tuples that are already ”seen” while discovering the top-h of the given query, in order to accelerate the query processing and subsequent getnext operations. The QR2-system also handles logic of parsing the returned elements into a format that Qr2 can handle. This means that parsing returned XML, JSON or parsing HTML DOM elements to get the data into required shape.

In addition to the query history, retained in the session variable, 1D-RERANK and MD-RERANK apply an on-the-fly indexing that detects the dense regions and proactively crawls top ranked tuples to save on processing future user queries. We use a MYSQL database to store the tuples in the dense region.

5.2 DBLoc Architecture

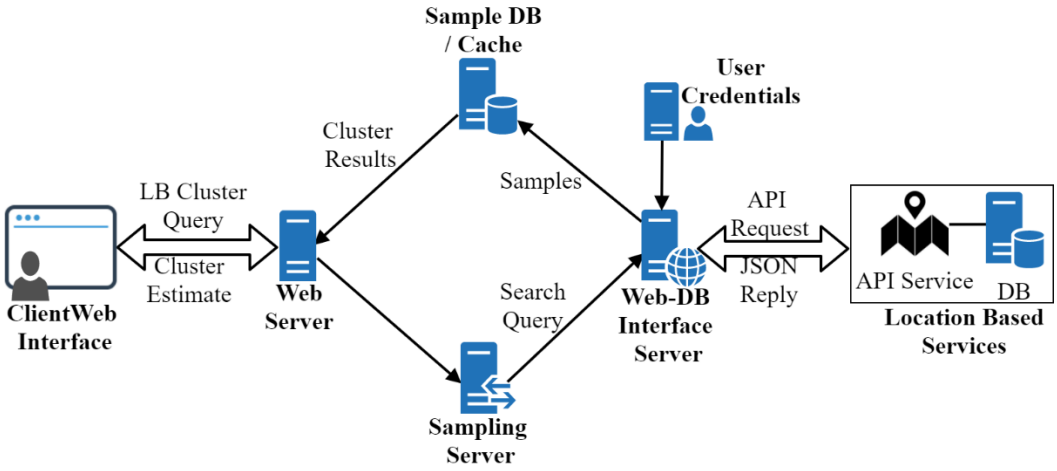


Figure 10 Architecture of DBLOC system

Figure 10 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. 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. 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. 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. 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.

Chapter 6

USER INTERFACE

6.1 QR2

6.1.1 Filtering and Ranking Section

Filtering section: The filtering section is used for specifying filtering predicates using a user-friendly web interface. This interface is common in many web databases, especially in Bluenile and Zillow. For Blue Nile, the user can adjust the price, carat, cut, color, and clarity sliders to search between a particular area and also select nominal attributes like shape of diamond. For Zillow, in addition to the location (e.g. the city and zip code), as shown in Figure 11(c) and (d), we include all the filtering conditions, such as number of bedrooms and price, in the filtering section.

Ranking section: For QR2, as a reranking service, the ranking section is special. The ranking section should provide a user-friendly way of identifying the user preference, even for the users that do not have an understanding of the ranking function notion. Obviously, expecting the user to compose a function for the query is not realistic. Hence, one of the challenges of this project was designing this section in a way that is convenient for the ordinary users. After investigating different alternatives, we designed the ranking section as follows:

_ 1D: Similar to the order by clause in a SQL query, for one-dimensional reranking, the user needs to simply specify the ranking attribute, as well the ordering direction, i.e., ascending or descending. As specified in Figure 11(e), in addition to the ranking attribute and the sorting direction, the user can specify the number of results per page.

_ MD: This component aims to provide a convenient way for the ordinary users to specify their preference. To do so, after normalizing the attribute domains, it uses a slider for each of attributes chosen for ranking. For each attribute A_i, the preference co-efficient, wi is specified by a slider value in the range of [-1,1]. Based on the slider values, the user-specified ranking function is mP i=1 wi:A_i. Figure 11(c) and Fig. 3(d) show two example instances of the MD ranking section for Blue Nile and Zillow, for the ranking functions price - 0.1 carat - 0.5 depth and price- 0.3 square feet, respectively. In addition to the slider, we also suggest a list of popular functions for the user to choose from.

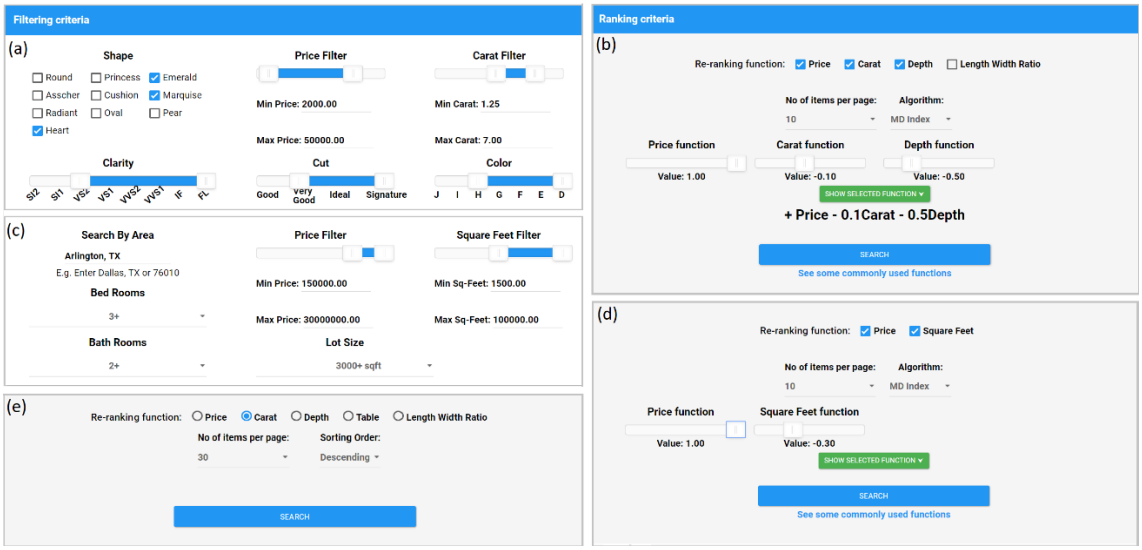
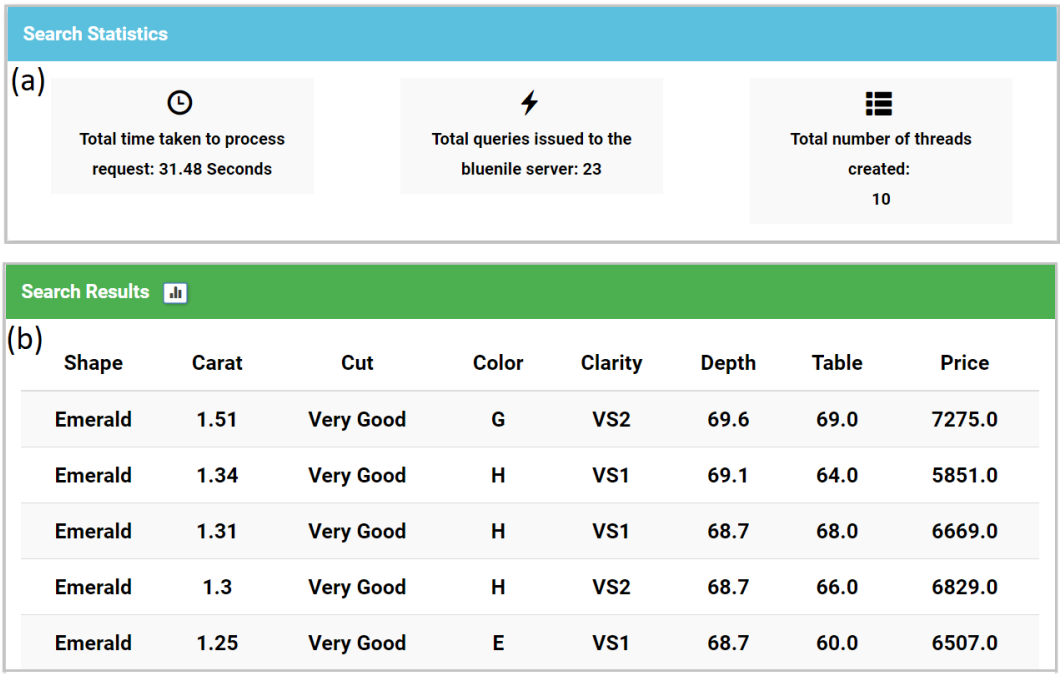


Figure 11 Input user interface of QR2

6.1.2 Output and Statistics

Once the query is issued, the system processes and returns the table of top-k results (k is specified by the user). Each row of the table shows the details of a tuple while clicking on it opens the web database page of the tuple. The get-next button allows the user to get the next page of results. Along with the results, the user is also provided with a small panel

providing statistics such as query cost in terms of the number of queries issued to the web database and processing time. Figure 12 shows a screen shot of results and the statistics for a reranking query on Blue Nile. Similarly, for Zillow and the ranking function Price - 0.3*Carat, the system issued 27 queries to the Zillow server, which took 33 seconds.



Shape	Carat	Cut	Color	Clarity	Depth	Table	Price
Emerald	1.51	Very Good	G	VS2	69.6	69.0	7275.0
Emerald	1.34	Very Good	H	VS1	69.1	64.0	5851.0
Emerald	1.31	Very Good	H	VS1	68.7	68.0	6669.0
Emerald	1.3	Very Good	H	VS2	68.7	66.0	6829.0
Emerald	1.25	Very Good	E	VS1	68.7	60.0	6507.0

Figure 12 Output interface of QR2, (a) shows search statistics of QR2 and (b) shows search results

6.2 DBLoc

6.2.1 Input Section

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 ϵ = 0.02,minPts = 10 and query budget = 300 is presented in Figure 13. [1]

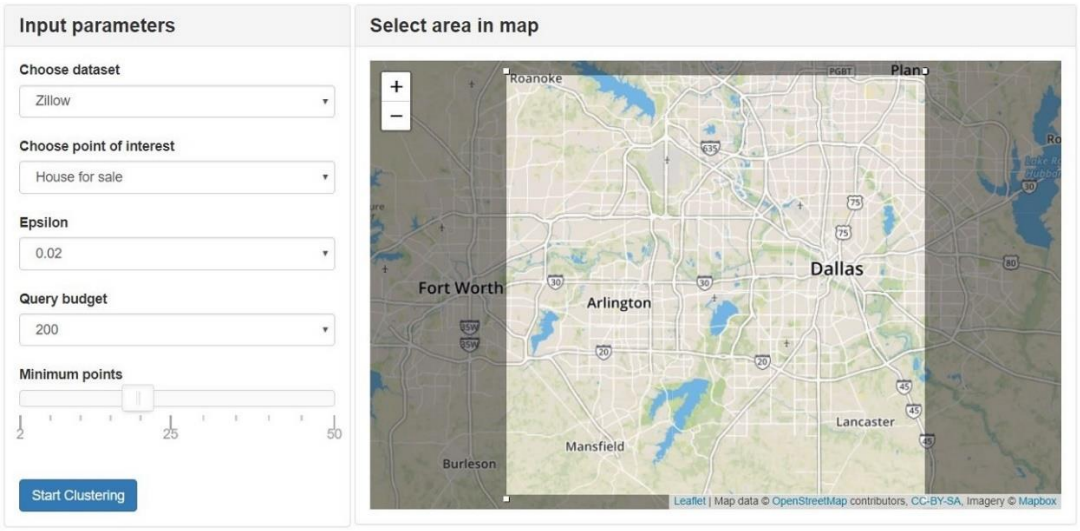


Figure 13 input parameters of DBLOC

6.2.2 Output Section and Statistics

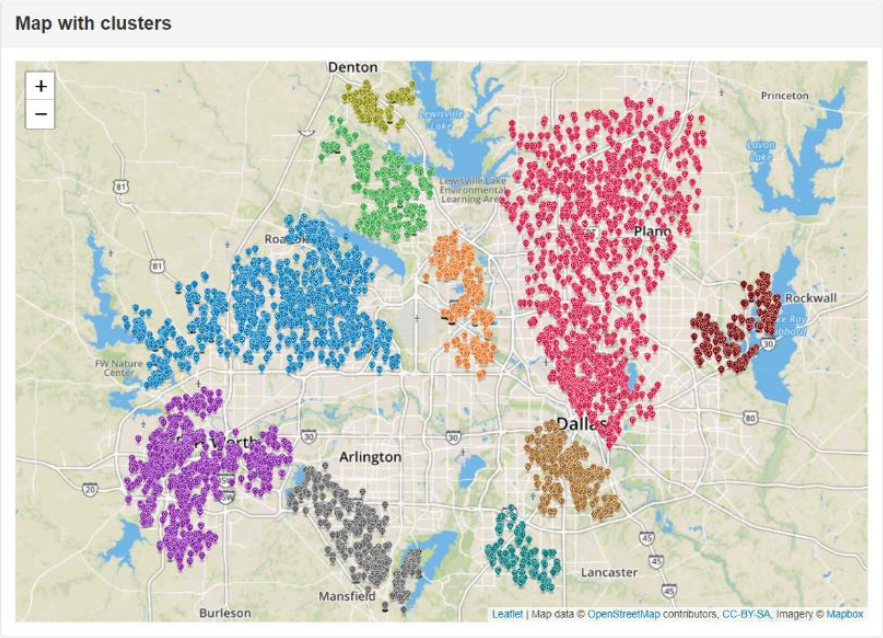


Figure 14 Output Map of DBLOC with cluster results

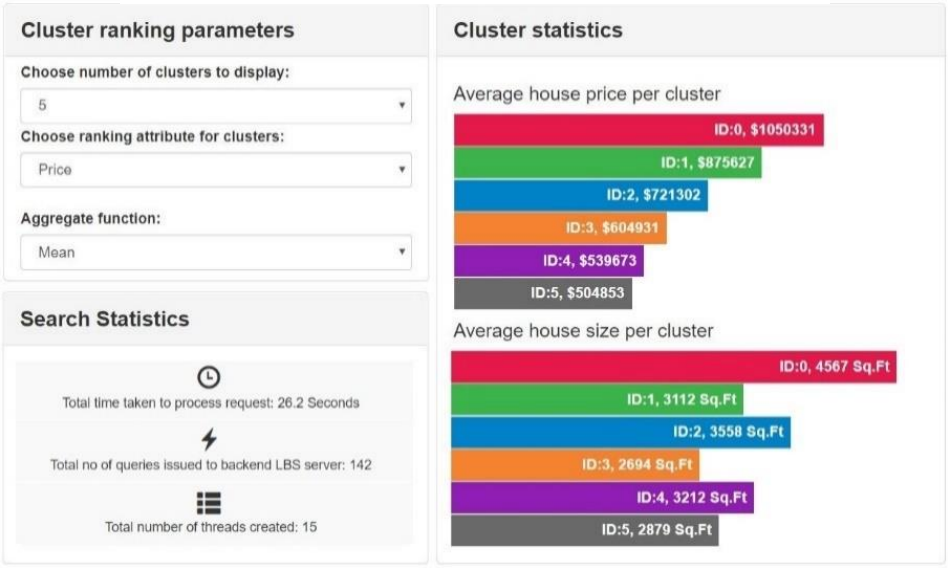


Figure 15 Cluster ranking parameters and cluster statistics

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 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 15). The users can set the value of k, choice of attribute and aggregate measure using the controls in “Cluster ranking parameters” panel. The “Cluster

Figure 16 Clusters compared with each other

comparison” panel (Figure 16) 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 kNN 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.

Chapter 7

CONCLUSION

7.1 Summary

We proposed to address two problems namely:

1. Retrieve single tuples from deep web databases according to user preferences
2. Get aggregate information from Deep web databases and analyze and present them to the user.

To achieve this goal, we have developed two systems QR2 and DBLOC systems, which are third party services that (1) Enable the on-the-fly processing of queries with any ranking function defined by the user to a web database, (2) Capture aggregate information over a group of tuples by clustering over Location based services.

QR2 uses nothing but the public search interface of the web database and addresses a wide range of user's preferences in ranking the results, even if not supported by the database.

DBLOC, enables analytics over the discovered clusters, thus helping the users to get more insight about the output. DBLOC does not assume full access to the underlying data and requires noting but limited access to kNN interface provided by the LBS.

More than just coming up with algorithms QR2 and DBLOC have also demonstrated how to develop a scalable and reliable solution which can be used when many users access such a service. Along with that we have also designed an efficient User interface which exactly captures the user input parameters and displays the output according to the user's need.

REFERENCES

[1] Y. D. Gunasekaran, A. Asudeh, S. Hasani, N. Zhang, A. Jaoua and G. Das, "QR2: A Third-party Query Reranking Service Over Web Databases," in *ICDE*, 2018.

[2] A. Abolfazl, N. Zhang and G. Das, "Query Reranking as a service," in *vol. 9, no. 11, pp. 888–899, VLDB*, 2016.

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

[4] B. Moon, H. V. Jagadish, C. Faloutsos and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," in *TKDE*, 2001.

[5] P. Xu and S. Tirthapura, "Optimality of clustering properties of spacefilling curves," in *TODS*, 2014.

[6] "wikipedia.org," [Online]. Available: https://en.wikipedia.org/wiki/Space-filling_curve.

[7] C. Sheng, N. Zhang, Y. Tao and X. Jin, "Optimal algorithms for crawling a hidden database in the web,"," in *VLDB*, 2012.

[8] Y. D. Gunasekaran, M. F. Rahman, S. Hasani, N. Zhang and G. Das, "DBLOC: Density Based Clustering over LOCation Based Services," in *SIGMOD*, 2018.

[9] M. F. Rahman, W. Liu, S. B. S. S. Thirumuruganathan, N. Zhang and G. Das, "Density based Clustering over Location Based services," in *ICDE, pages 461-472*, 2017.

Biographical Information

Yeshwanth is a Masters student at UT Arlington specializing in Data exploration/Data science and software engineering. As a member of DB Exploration Lab, Yeshwanth worked under Dr. Gautam Das in research areas which include databases, information retrieval, query processing and hidden web databases.

Yeshwanth is also as a Software Eng. / Data Analyst Intern at CareCognitics, where he builds exciting products and data models to help doctors provide personalized care to their patients. Prior to his masters, Yeshwanth has worked at TCS, where his team was responsible for building and delivering cloud solutions for TCS's customers. He was also responsible for building and configuring data-center networks, servers and storage.