# GroupFinder: A New Approach to Top-K Point-of-Interest Group Retrieval

Kenneth S. Bøgh
Department of Computer Science
Aarhus University
u071354@cs.au.dk

Anders Skovsgaard
Department of Computer Science
Aarhus University
anderssk@cs.au.dk

Christian S. Jensen
Department of Computer Science
Aarhus University
csj@cs.au.dk

## ABSTRACT

The notion of point-of-interest (PoI) has existed since paper road maps began to include markings of useful places such as gas stations, hotels, and tourist attractions. With the introduction of geo-positioned mobile devices such as smartphones and mapping services such as Google Maps, the retrieval of PoIs relevant to a user's intent has became a problem of automated spatio-textual information retrieval. Over the last several years, substantial research has gone into the invention of functionality and efficient implementations for retrieving nearby PoIs. However, with a couple of exceptions existing proposals retrieve results at single-PoI granularity. We assume that a mobile device user issues queries consisting of keywords and an automatically supplied geo-position, and we target the common case where the user wishes to find nearby groups of PoIs that are relevant to the keywords. Such groups are relevant to users who wish to conveniently explore several options before making a decision such as to purchase a specific product. Specifically, we demonstrate a practical proposal for finding top-$k$ PoI groups in response to a query. We show how problem parameter settings can be mapped to options that are meaningful to users. Further, although this kind of functionality is prone to combinatorial explosion, we will demonstrate that the functionality can be supported efficiently in practical settings.

## 1. INTRODUCTION

Many online services exist that support the retrieval of points-of-interest (POIs), including international services such as Google Maps [3] and country-specific services such as the Danish Krak [4]. In addition, vehicle navigation services offer category-based PoI search functionality bases on the user's current location. Services such as these retrieve nearest-neighbor PoIs relative to the user's location. Further, they do this at single-PoI granularity, meaning that they consider each PoI in isolation and return results consisting of $k$ single PoIs, each of which is a result object. In many use cases, this functionality is what users expect. For example, this can occur when a driver is looking for the most convenient gas station in order to re-fuel the vehicle. In contrast, we support the common browsing behavior use case, where users prefer to explore several options prior to making a decision. Such behavior can occur when a tourist is looking for a place to get dinner. In this case, the user may wish to look at the menu options and prices at several restaurants prior to making a dining decision, and seating availability may cause the user to consider additional options. Here, the standard $k$ nearest neighbor functionality falls short: While the nearest neighbor is close to the user's location, it may be relatively far from the next nearest neighbor, which may again be relatively far from the third nearest neighbor. In the above browsing behavior case, the user is better served by a group of relevant PoIs that are both near the user's location and near each other. We support the retrieval of $k$ such groups. To improve the utility of single-object granularity results, previous research has tried to incorporate scoring functions to take the quality of each PoI into account [6], thus returning the best PoIs in terms of both distance and quality. While this does hold the potential for retrieving better results, they still have single-object granularity. For example, a nearby, high-quality restaurant may be too expensive, and the next nearest neighbor may still be relative far away. One study considers two instance of a collective query that treat a set of PoIs as a single object [1, 2]. Both return only a single set (i.e., $k = 1$), and both apply Boolean matching to the keyword part of the queries and apply only ranking to the spatial parts of the queries. These queries support the different kind of use case where the PoIs in a result set complement each other and where a user wishes to visit every single PoI in the result set. In our use case, the PoIs in a result compete against each other, and the user intuitively wishes to find the best one. We demonstrate GroupFinder, a web-based service that returns top-$k$ groups of PoIs according to a scoring function. In addition to taking into account the user's current location and user-supplied query keywords, the function takes into account transportation-mode information that captures how the user intends to travel to a group of PoIs and how the user intends to travel between the PoIs in a group. Groups are retrieved by means of a new type of index called a Group-Extended R-tree (GER-tree). Query results are displayed using the Google Maps infrastructure, and the responsive web design front end Twitter Bootstrap is leveraged in order to make the service equally useful on traditional desktops and new devices such as tablets and smartphones. The remainder of the paper is organized as follows. Section 2 introduces the indexing technique that is the basis for GroupFinder. Section 3 presents an overview of the communication and architecture of GroupFinder. Then Section 4 presents the user interface and describes how results are displayed. Finally, Section 5 details the demonstration setup.

*Proceedings of the VLDB Endowment, Vol. 6, No. 12*
*Copyright 2013 VLDB Endowment 2150-8097/13/10... $ 10.00.*

## 2. PRELIMINARIES

GroupFinder exploits a new indexing technique, the GER-tree, that indexes PoIs with a geo-location and a text description. First, a vocabulary of searchable keywords is built based on the text descriptions of the PoIs. For each such word, an R-tree is built on the location of each PoI that contains the word in its text description. This yields a forest of R-trees that each enable spatial queries against the PoIs with descriptions that contain a particular word [5]. Next, to support the retrieval of groups rather than single objects, each R-tree is extended with purposefully designed, compressed histograms in all non-leaf nodes. The histogram in a node contains information on how densely grouped the subtrees of the node are, how many objects the groups contain, and give the minimum bounding rectangles of the subtrees. Further, a score is calculated for each PoI in a tree, to indicate its relevance to the tree's keyword. These scores are averaged for groups and added to the compressed histograms. The resulting index enables efficient retrieval of groups in the vicinity of a location.

## 3. SYSTEM ARCHITECTURE

GroupFinder is completely web based and is thus accessible from any web browser. To provide the best user experience, we employ a client-based approach to retrieve the current location of the user, while we use the conventional client/server approach to process queries.

### 3.1 Geo-Locating a User

We apply several techniques to accurately and non-invasively geo-locate users in a generic, platform-independent manner.

We first use the ClientLocation tool of the Google Maps infrastructure to geo-locate a user based on the user's IP. This approach allows us to identify the current city of the user in many cases, thus providing a rough estimate of the user's geo-location. When the user's approximate location is found using this approach, we use the location to position the map at the user's location at an overview zoom level.

Next, we ask the user for permission to use the HTML5 geolocation API. Having obtained this permission, we apply a finer zoom level to the map to better display the user's more accurate location. It is also possible for the user to indicate a preferred position by simply clicking on the map. This functionality adds flexibility and can be used to correct positioning errors.

Since modern browsers, including those of tablets and smartphones, support the geolocation API, we can always locate the user if given permission. We do not need to use any native apps that may be available on different mobile platforms. The resulting device independence is very attractive. The approach taken also allows us to provide users with a good experience while affording the users control of whether or not to reveal their actual location.

### 3.2 Client-Side Querying

To explain the process of retrieving the $k$ most relevant groups, we consider the case where a user is standing at the central railway station in the city of Aarhus, Denmark. This location is shown in Figure 1.

#### 3.2.1 *Keyword Handling*

To query for groups of PoIs, the user types a query keyword. In particular, when the web service is loaded, the available keywords are generated server side and sent to the client so that the client has an up-to-date set of keywords. When the user starts to type, Google-style suggestions are presented to make it as easy as possible for the user to formulate a query.

In the demonstration, we use a dataset derived from Google Places, since this is one of the best sources of PoIs. In Figure 2, the user searches for restaurants.

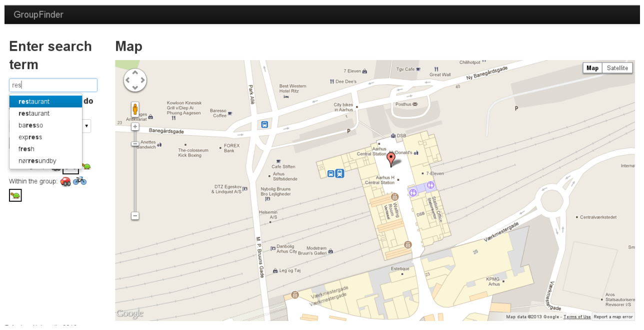

**Figure 1: Before Querying**

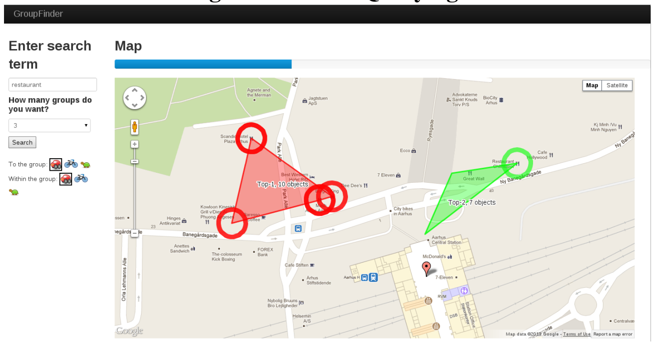

**Figure 2: A Query In Progress**

#### 3.2.2 *Distances To and Within Groups*

A user may prefer different transportation modes when traveling from the current location to a group. Likewise, a user may prefer different transportation modes when traveling between the PoIs in a group. For both types of travel, GroupFinder allows a choice among three transportation modes: by car, by bicycle, by foot. These settings are used to set problem parameters in the ranking function that control which groups are returned to the user. Specifically, when scoring a group, the ranking function takes into account the distance to the group and the group's density, and both aspects have a weighting parameter that takes a value in the range $[0, 1]$. Each transportation mode is then mapped to a value for the two parameters. The result is that nearby or dense groups are preferred when the turtle icon (by foot) is selected for the distance to groups and the intra-group distance. If the car icon is selected, spatial proximity and density are less important, and preference is instead given to groups with high average text relevance to the query and with many PoIs. In the query covered in Figure 3, the car icon has been chosen for both distances. Consequently, the result contains relatively large groups with textually very relevant objects, at the expense that one group being relatively far from the query location.

#### 3.2.3 *The Number of Groups*

While the underlying techniques allow the retrieval of any number $k$ of groups, GroupFinder allows the user to retrieve from 1 to 5 groups. We expect that if a user is facing a light-weight decision (e.g., buy milk), one nearest group is enough (e.g., to find a store that is open and has milk), while the user may want to see several groups if facing a more "complex" decision (e.g., finding a restaurant to have dinner at). As the underlying techniques return groups progressively, the service displays groups as soon as they are available. Figure 2 shows a query in progress: two groups have been found, and a blue progress bar above the map indicates the percentage of the query that has been completed.

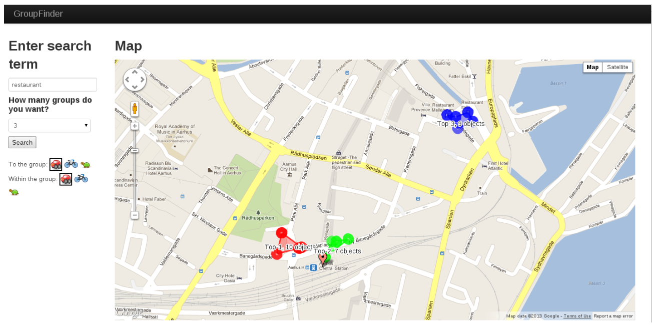

**Figure 3: Query Result**

## 3.3 Query Processing

Query processing as seen from the client is outlined in Algorithm 1. To maximize the amount of concurrent users, all communication is asynchronous. The first request in Line 3 simply sends the collected information to the server that responds with a $searchid$. The client then repeatedly requests additional groups with a small delay between requests until it has the desired number of groups. While several factors have an influence on the query time, a query typically runs at 300-1500 milliseconds on an Intel(R) Core(TM) i5-2520M CPU at 2.50Ghz. To offload the server, a group only contains a set of coordinate pairs, while the bounding region and the pan and zoom instructions are calculated on the client using JavaScript. Information about the individual PoIs in a group is obtained by a separate call when the user clicks on the group. On the server side, the web service is implemented using a Tomcat server and Java. The indexing and query processing are also implemented in Java, and they are deployed directly in the server to minimize response times. When a query is initiated, a new thread is started to query the index, and a mapping between the thread and $searchId$ is created. The query thread finds the tree to query and initiates a search in the tree. The threads that query the same tree share the same buffer. Thus, different threads that access the same nodes benefit from the shared buffer and save disk reads, which improves performance. When the client requests the next group, the server simply looks up the thread handling the query in the map and checks if new groups are available. If so, the top group is returned to the client.

**Algorithm 1**: Search from client

```
1  begin
2      $amountRecieved \leftarrow 0$;
3      $searchID \leftarrow$ HTTP GET:
       $[lat, lng, term, amount, distIGR, distTGR]$;
4      while $amountRecieved < amount$ do
5          $group \leftarrow$ HTTP GET: $[searchID]$;
6          if $group! = NULL$ then
7              $amountRecieved++$;
8              $PanAndZoom(group)$;
9              $Hull \leftarrow ConvexHull(group)$;
10             $DisplayGroup(Hull, group)$;
11 end
```

## 4. THE GROUPFINDER WEB SERVICE

We proceed to cover the web service prototype and its platform independence.

## 4.1 User Interface

The user interface consists of two parts—see Figure 4. On the left side, the user can provide the query parameters, as described in Section 3.2. We have chosen to use discrete options for the travel preferences. These options map well to different transportation modes, and this approach makes it easy to obtain query results.

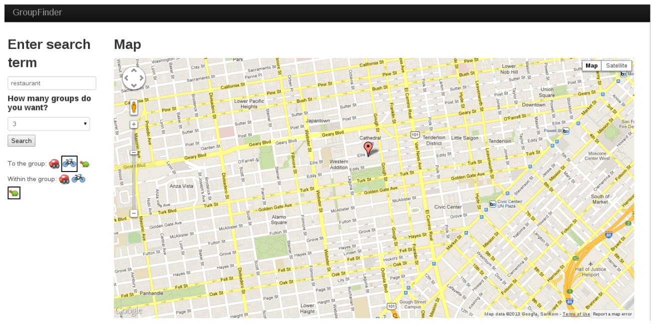

**Figure 4: GroupFinder Interface, Desktop Version**

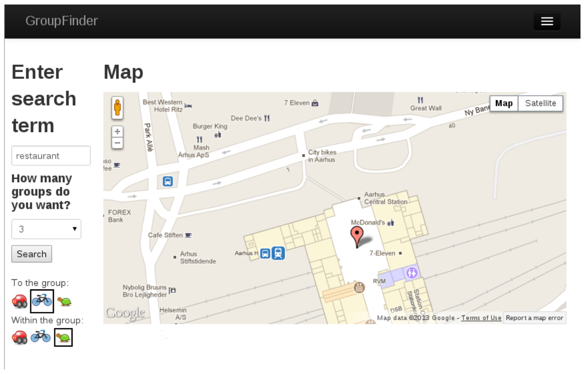

**Figure 5: GroupFinder Interface, Tablet Version**

The fields for keywords and the number of groups are self explanatory. It should be mentioned that the quality of the groups found is independent of the number of groups to be retrieved. The best group is found solely based on the keyword and distance parameters. On the right side, we use Google Maps to let the user choose their location if the user wants to use a location different from the one found automatically.

## 4.2 Querying

Once a query has been issued, the client side scripts start to request results from the server. When the first group is received, it is displayed on the map by zooming and panning so that the user's location and the group fit on the map. The pan and zoom adjustment of the map is done in a smooth animation, so that the user does not loose track of their location. When the next group is received, we again zoom and pan to display the user's location and the two groups on the map. This process terminates when all groups have been presented. A progress bar indicates the degree of completion and disappears when the search is complete. Figures 1–3 depict the process of searching for three groups of restaurants near the central train station in the city of Aarhus, Denmark.

## 4.3 Result Presentation

The results are returned and displayed as groups. To clearly outline a group, we calculate the convex hull of the group on the client

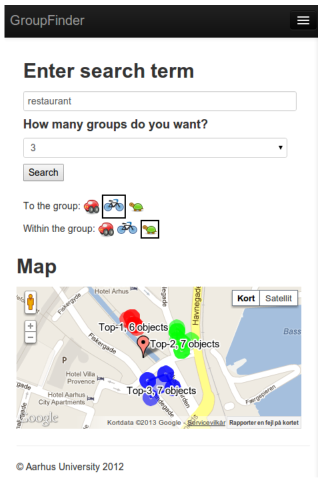

Figure 6: GroupFinder Interface, Smartphone Version

and then color the polygon that represents the convex hull. This is seen in Figures 2 and 3. Each vertex of the convex hull is marked by a circle to ensure that the group is always visible, even when the objects in the group lie on a line. We overlay a text that describes each of the groups to help the user decide which group is most desirable. The text includes the rank of the group in the list of all results along with the number of objects in the group. Since a group can contain many objects, it can be difficult to see how many objects a group actually contains when using normal maps markers. To provide at better overview, we have replaced the normal markers with rings. This ensures that objects close to each other can be told apart. If a user clicks on a group, the map automatically pans and zooms onto that group while keeping the user's location on the map. That way, the user can get a better look at a specific group without losing track of where they are in relation to the group. Another click on the group makes the map zoom back out to again show all result groups.

## 4.4 Effect of Distance Settings

To show the impact of the distance settings, Figure 7 shows the result of the query considered so far (shown in Figure 3), but with the distance indicators set at the other end of the scale. The groups

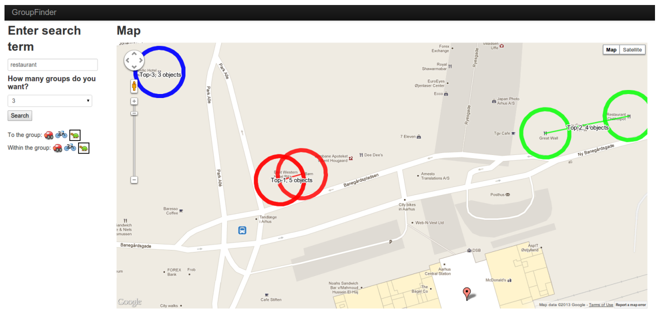

Figure 7: Effect of Distance Settings

found now are more dense, contain fewer PoIs, and are closer to the query location. The result matches the intuitive semantics well, since a person on foot is unlikely to want to walk far to visit a group where the objects are also not in close proximity of each other. This result is "opposite" to the result in Figure 3, where the groups contain more objects, are further away from the query location, and contain objects that are further apart from each other. Although we are aiming to support browsing behaviour, by providing groups of PoIs we do not ignore single PoIs. As such, if there is a very relevant PoI close to the user, but not close to other PoIs it may be returned as a group containing a single object.

## 4.5 Cross Platform Support

As mentioned, we use the Twitter Bootstrap front end framework to display our data. This allows for the same interface to be used across different client platforms without the need for any native apps on special devices such as tablets or smartphones.

Thus, a user is always presented with the same, familiar interface when using GroupFinder from different platforms. Figure 4 shows the desktop interface, while Figure 5 shows the interface on a tablet, and Figure 6 displays a possible smartphone look at the interface, although orientation and device resolution can affect the experience.

The responsive design of Twitter Bootstrap moves the individual parts of the interface around, but it is still easy to find the buttons needed, thus making the web service useful across platforms.

## 5. DEMONSTRATION

GroupFinder offers new functionality for PoI querying. This functionality targets use cases characterized by browsing behavior where users wish to explore several competing PoIs prior to making a decision such as choosing a restaurant for dinner or choosing which shoes to buy. We are not aware of any other services that target this behavior.

The demonstration is done in two parts. First, we introduce the underlying indexing and query processing techniques in more detail, showcasing how we rank groups of PoIs and how the query parameters affect the query processing.

Second, we give the participants the chance to experience the system themselves in a live demonstration with a publicly available web service. We provide data for Aarhus and other cities to make the demonstration as relevant as possible. The attendees are able to chose from a huge set of keywords and will be able to quickly get an overview of where in their vicinity to go for food, entertainment, or a drink.

The web based interface allows users to use the web service regardless of operating system, device hardware, or browser preference. This demonstration therefore allows the attendees to not only experience GroupFinder during the demonstration sessions. They can also use GroupFinder on their own at any time to explore the surroundings of the conference location.

## 6. REFERENCES

[1] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective Spatial Keyword Querying. In *Proc. SIGMOD*, pp. 373–384, 2011.

[2] X. Cao, G. Cong, C. S. Jensen, J. J. Ng, B. C. Ooi, N.-T. Phan, and D. Wu. SWORS: A System for the Efficient Retrieval of Relevant Spatial Web Objects. *PVLDB*, 5(12):1914–1917, August 2012.

[3] Google. Google Maps. `https://maps.google.com/`.

[4] Krak. Krak. `http://www.krak.dk/`.

[5] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient Processing of Top-k Spatial Keyword Queries. In *Proc. SSTD*, pp. 205–222, 2011.

[6] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient Processing of Top-k Spatial Preference Queries. *PVLDB*, 4(2):93–104, November 2010.