Seed-Driven Geo-Social Data Extraction

Isaj, Suela; Pedersen, Torben Bach

Published in:
International Symposium on Spatial and Temporal Databases

DOI (link to publication from Publisher):
10.1145/3340964.3340973

Publication date:
2019

Link to publication from Aalborg University

Citation for published version (APA):
Seed-Driven Geo-Social Data Extraction

Suela Isaj
Aalborg University
suela@cs.aau.dk

Torben Bach Pedersen
Aalborg University
tbp@cs.aau.dk

ABSTRACT
Geo-social data has been an attractive source for a variety of problems such as mining mobility patterns, link prediction, location recommendation, and influence maximization. However, new geo-social data is increasingly unavailable and suffers several limitations. In this paper, we aim to remedy the problem of effective data extraction from geo-social data sources. We first identify the limitations of extracting geo-social data. To overcome the limitations, we propose a novel seed-driven approach that uses the points of one source as the seed to feed as queries for the others. We additionally handle differences between, and dynamics within the sources by proposing three variants for optimizing search radius. Furthermore, we provide an optimization based on recursive clustering to minimize the number of requests and an adaptive procedure to learn the specific data distribution of each source. Our comprehensive experiments with six popular sources show that our seed-driven approach yields up to 14.3 times more data overall, while our request-optimized algorithm retrieves up to 95% of the data with less than 16% of the requests. Thus, our proposed seed-driven approach set new standards for effective and efficient extraction of geo-social data.

CCS CONCEPTS
• Information systems → Spatial-temporal systems; Information extraction.

1 INTRODUCTION
Each year social networks experience a continuous growth of 13% in the number of users (http://wearesocial.com/uk/blog/2018/01/global-digital-report-2018). Consequently, more and more information is available regarding the activity that the users share, events in which they participate and the new connections they make. When data collected by social networks contain social connections (friendship links, mentions, and tags in posts, etc) as well as geographic information (check-ins, geo-data in posts and implicit location detection), then this data is usually referred as geo-social data. Geo-social data have attracted studies regarding location prediction, location recommendation, location-based advertisement, urban behavior, etc. The primary sources of geo-social data are location-based social networks (LBSNs) such as Gowalla, Brightkite, and Foursquare, which contain social ties, check-ins, tips and detailed information about locations. However, Gowalla and Brightkite were closed in 2012, whereas Foursquare has blocked the extraction of check-ins from its API (Application Programming Interface - set of functions and procedures that allow data extraction from a source). Other secondary sources of geo-social data are social networks such as Facebook, Twitter, Flickr, etc. Social networks are characterized by richness and variety of data, making them an attractive source for data extraction. However, the percentage of geo-located posts reported in the literature is less than 1% ([5, 15, 23]). Furthermore, they provide rich information about users, their networks, their activities but only a few details about locations (only the coordinates). Another less common source of location data (not necessarily geo-social) are directories such as Yelp, Google Places, TripAdvisor, etc, which contain locations with details such as name, phone, type of business, etc and sometimes accompanied by user reviews. In the majority of the cases, directories do not contain user profiles; even when they do, the API does not provide functions to extract user’s information. Hence, it is necessary to use several sources in order to gain a complete dataset of geo-social data.

Not only is geo-social data scattered over several sources but the APIs of the sources are also highly restrictive regarding the number of requests, the amount and the type of data that can be extracted, etc. Instead of extracting the data, publicly available datasets can be used. However, their usability is limited because sometimes they lack the details about users’ profiles or the locations, which could be of interest for the research purpose. Besides, the check-ins/photos/posts/reviews are sparse and scattered all over the globe, affecting the quality of the experiments while mining frequent patterns, mobility patterns, urban behavior, etc. Enriching these datasets with the missing details is not possible because the data is anonymized, so the link with the source is lost. Even when the data is not anonymized, the datasets are old (2008-2013) and they can not map to the existing users or locations of nowadays. When we analyzed 32 papers from 2009 to 2018 using geo-social data, we found that no less than 50% used datasets that are 3-8 years older than the published article (see Appendix A in [14]).

To sum up, geo-social data is becoming even more needed and even less accessible. We thus, address the problem of location-based geo-social data extraction from social networks and location-based sources. We introduce the limitations of six sources of geo-social data, namely: Flickr, Twitter, Foursquare, Google Places, Yelp, and Krak. Then, we propose a seed-driven algorithm that uses the points of the richest source (the seed) to extract data from the others. Later, we introduce techniques to optimize the selection of radius and seed points. Our main contributions are: (i) We provide an analysis of the current limitations of data extraction from six popular geo-social data sources. (ii) We identify and formulate the problem of maximizing the extracted geo-social data while minimizing the requests to the sources. To the best of our knowledge, we are the first to optimize the data extraction process in social networks and location-based sources. (iii) We propose a novel algorithm for data
extraction that uses the points of one source as seed to the API requests of the others. Our seed-driven algorithm retrieves up to 98 times more data than the default API querying. (iv) We introduce an optimized version of our algorithm that minimizes the requests and ensures maximized data extraction by recursively adapting the radius and the centroid of the query region. We retrieve around 90% of the data using less than 16% of the requests.

The remainder of the paper is structured as follows: first, we describe the related work in Section 2; then, we introduce the definitions and the data extraction problem in Section 3; later, we categorize the limitations of the data extraction process and we provide preliminary results from six sources in Section 4; we continue with formalizing our proposed algorithm in Section 5; next, we test the proposed solutions through real-time querying of the sources and we compare the results in Section 6; and finally, we conclude and provide further insights on our work in Section 7.

2 RELATED WORK
Despite the growing interest in geo-social related topics, the existing related work does not focus specifically on optimizing the data extraction process. Most of the existing research uses either publicly available datasets [4, 7, 9, 10, 21, 22, 29, 30, 33, 35, 36, 38–40], crawl using the default settings of the API [3, 6, 11, 16, 24, 27, 32, 34, 37] or do both [18, 25]. The (sparingly described) crawling methods used in these papers can be categorized as either user-based crawling, location-based crawling or keyword-based querying.

User-based crawling. User-based crawling is based on querying users for their data and their networks as well. A user-based crawling technique mentioned in several studies is the Snowball technique [12, 31]. Snowball requires a prior seed of users to start with and then, traverses the network while extracting data from the network of friends. Nonetheless, Snowball is biased to the high degree nodes [20] and requires a well-selected seed. Another interesting method is to track the users that post with linked accounts [1, 13, 28], for instance, users posting from Twitter using the check-in feature of Foursquare. Nevertheless, this method is limited only to linked accounts, whose percentage is less than 1%.

Location-based crawling. Location-based crawling requires no prior knowledge, and the extraction process can start at any time. It is based on extracting data near or within a specific area. Lee et al. [19] use a periodical querying based on points extracted from Twitter. First, Twitter is queried for initial points. Then, in a later step, other requests are performed using the initial points as query points, focusing on areas detected by the users. Thus, in each step n, the points discovered in step n − 1 are used to perform the new queries. We will refer to this method as Self-seed.

Keyword-based querying. As the name suggests, the source is queried [not necessarily crawled] with a keyword to find relevant data. The keyword-based querying is widely used by the research on topic mining, opinion mining, the reputation of entities, quality of samples and several related topics [2, 17] but not for geo-social topics since querying with a keyword does not guarantee that the retrieved data will be located in the queried location. For example, querying Twitter with the keyword ‘Brussels’ can return tweets in Brussels, tweets talking about Brussels but not located there, and even tweets about brussels sprouts.

Discussion. Obviously, the keyword-based querying is not of interest due to the noise it brings. The user-based crawling requires prior information about a seed of users and applies only to social networks. Subsequently, it leaves aside other location-based sources such as Yelp, TripAdvisor, and Google Places. Moreover, if the study is based on a region of interest, the user-based crawling results in a lot of irrelevant data because even if the seed of users is well-selected from the region of interest, there is no guarantee that the friends will check-in in the area of interest. Consequently, user-based crawling produces wasted requests. The method described by [19] has some similarities with ours because it is location-based crawling and focuses on performing requests on areas discovered previously. In comparison, our approach differs significantly because (i) instead of selecting points from a single source and querying itself, we use a seed source to query multiple sources, (ii) we minimize the number of requests performed while maximizing the data extracted (iii) our seed-driven data extraction approach does not need periodical querying; it can be run continuously and simultaneously for all the sources, resulting in faster data extraction process, compared to several months like in [19]. To sum up, our data extraction approach is faster, richer, request-economic, and includes multiple sources.

3 PROBLEM DEFINITION
The notion that we will use widely in the paper is a location. A location in a directory is a venue with a geographical point and additional attributes like name, category, etc. Social networks contain activities such as check-ins, tips, photos and tweets which are geo-tagged. We denote the locations associated with the activities as derived locations and for brevity, just as locations.

Definition 1. A location \( l \) is a spatial entity identified within the source by a unique identifier \( id(l) \). A location \( l \) has a set of attributes \( A = \{a_1, a_2, ... a_n\} \) accompanied by their values \( \{a_1(l), a_2(l), ... a_n(l)\} \). A required attribute for a location \( l \) is its geographical coordinates denoted as \( p(l) \)

For example, \( l_1 \) is a tweet with id = 1234567, where \( A = \{text, user, point\} \) and the values are \{‘Nice day in park’, 58302, <57.04, 9.91>\}. Geo-social data sources usually offer an Application Programming Interface (API), which is a set of functions and procedures that allow accessing a source to obtain its data. Location-based API calls allow querying with (i) a point \( p \) and a radius \( r \), (ii) a box \( <p_1, p_2, p_3, p_4> \) and (iii) keywords. We will not consider keyword-based querying due to the noise it brings (see Section 2). The circular querying and the rectangular querying are quite similar as long as the parameters are the same. In this work, we use querying with a point \( p \) and a radius \( r \) and refer to the searched area as \( Circle(p, r) \). We define a geo-social data source formally as:

Definition 2. A geo-social data source \( S \), short as source, consists of the (complete) set of locations \( L(S) \) and a source-specific extraction function \( API: P \times R^+ \rightarrow 2^{L(S)} \), where \( P \) is the domain of geographical points and \( R^+ \) is the domain of non-negative numbers. \( API(p, r) \) queries with a centroid \( p \in P \) and a radius \( r \in R \) and returns a sample of locations \( L^*_{p, r} \), such that for each \( l \in L^*_{p, r} \), \( p(l) \in Circle(p, r) \) and \( |L^*_{p, r}| \leq M_S \), where \( M_S \) is the maximal result size for \( S \).
If $S$ is Twitter, then $L(S)$ is the complete set of tweets (all the tweets posted ever on Twitter). A request with a point and a radius $< p, r >$ will retrieve a sample $L^p_r$ of size at most $M^p_S$ of the underlying activities in Circle$(p, r)$. So, if $M^p_S = 100$, then $L^p_r \leq 100$ locations. Given that the requests are limited, they need to be used wisely in order to retrieve the largest combined result size. For example, if the first request retrieves the locations $\{l_1, l_2, l_3, l_4\}$ and the second request retrieves $\{l_2, l_4, l_5, l_6\}$, then the second request contributed with only one new location ($l_5$).

**Problem definition.** Optimizing geo-social data extraction is the problem that given a source $S_i$ and a number of requests $n$ finds the sequence of pairs of point and radius $\{< p_1, r_1 >, < p_2, r_2 > \ldots < p_n, r_n >\}$ such that the size of $L_i = \bigcup_{j=1}^{n} L^p_{r_j}$ is maximized.

The problem aims to obtain a good compromise between the number of requests and the number of locations $L_i$. The optimal solution is a combination of $\{< p_1^*, r_1^* >, < p_2^*, r_2^* > \ldots < p_n^*, r_n^* >\}$ such that $L_i$ is maximal (the optimal $L^*_i$). Given the API limitations, trying exhaustively all possible values and combinations of $p$ and $r$ to find $L^*_i$ is not feasible. Hence, we propose solutions that are based on heuristics and assumptions. Before proposing our solutions, let us first introduce the API limitations for each source.

## 4 LIMITATIONS OF EXISTING GEO-SOCIAL DATA SOURCES

With regard to quantifying the limitations, we present preliminary results from querying six sources: Twitter, Flickr, Foursquare, Yelp, Google Places, and Krak. Krak is a Danish website that offers information about companies, telephone numbers, etc. In addition, Krak is part of Eniro Danmark A / S, which takes care of publishing The Yellow Pages. We queried all the sources simultaneously for the region of North Denmark during November-December 2017. With respect to gaining more data, we performed additional requests using different keywords (‘restaurant”, “library”, “cozy”, etc) as well as coordinates of the cities and towns in the region.

The API bandwidth refers to the number of requests allowed within a time frame. For example, Twitter allows 180 requests in 15 min; meanwhile, Krak has a time window of a month. Google Places allowed 1000 requests in a day before June 2018, and now, just one request per day. If more requests are needed, the cost is 0.7 US cents/request (first 200 USD free). In our data extraction and experiments, we fix the bandwidth of Google Places to 1000 requests in a day (the former default). The maximal result size is the maximal number of results returned by a single request. For instance, an API call in Flickr retrieves 250 photos, but in Google Places retrieves only 20 places. The historical access is related to how accessible the earlier activity is. Directories such as Yelp, Krak, and Google Places do not provide historical data; they only keep track of the current state of their locations. Foursquare provides only the current state of its venues and historical access to photos and tips by querying with venues. Flickr can go back in old photos, whereas Twitter limits the results only to the last couple of weeks. The supplemental results are data which do not belong in the query region, but they are still added to the API result. For example, if we search for “Zara” shop in city X, the API might return the “Zara” shop which is the closest to X but in city Y. We noticed that supplemental results are present only in directories, which aim to advertise and provide results anytime. Having access to the complete dataset means that the API can query the whole dataset, not just a sample. For example, the Twitter API accesses only a sample of 1% while others enable access to its complete dataset. Most social networks and directories offer free APIs at no costs. They also offer premium or enterprise services with monthly payment or pay-as-you-go services. While some sources have predefined pricing plans (Twitter, Foursquare, and Google Places), others offer the possibility to discuss the needs and the price (Yelp). Even though a premium service has fewer restrictions, it is still needed to keep costs down.

A summary of the limitations of social networks is presented in Table 1. Krak is restrictive with the bandwidth. Google Places has a very small result size and only one request per day. Flickr is promising in terms of the API limitations, while Twitter shows severe problems regarding the limitation to access historical data. Foursquare and Yelp could be considered similar in terms or limitations. The number of locations and the number of points for each source are presented in Table 2. Krak has a leading position with almost two orders of magnitude more results than any other source, followed by Flickr, Foursquare, Yelp, Google Places, and finally, Twitter. As for the temporal density, we recorded that in Flickr there are around 17 photos per day, in Twitter 10 photos per day, in Foursquare 0.03 tips and 0.36 photos per day (See Subsection 4.2 in [14] for more details on data scarcity). To sum up, a single source queried with the default API cannot provide a rich enough dataset. In the next section, we propose a novel algorithm that uses one of the sources as seed to extract data from the others and is capable of obtaining up to 14.3 times more data than single source initial querying (Section 6.1).

## 5 MULTI-SOURCE SEED-DRIVEN APPROACH

Section 4 studied the limitations of data extraction and quantified the performance of each of the sources. In this section, we propose a main algorithm and several adaptions to it that lead to an effective data extraction process.

### 5.1 Multi-Source Seed-Driven Algorithm

We will refer to the initial default API queries as Source Initial (SI) and to the set of locations they retrieve as $L_I$. Having no prior knowledge of the underlying data $L(S_I)$ makes it challenging to choose which API calls to perform. However, all the sources operating in the same region contain data that maps to the same physical world. For example, if there is a bar in the point (56.89 9.21) in Krak, probably around this point there might be this and other locations in Yelp, Google Places, and Foursquare and even some activity such as tweets, photos or check-ins in Twitter, Flickr, and Foursquare. This means that if the SI of a source is rich in terms of locations, then its knowledge can be used to improve the data extraction of the other sources. Hence, we propose a seed-driven approach to extract locations from multiple sources. The main idea is simple: selecting one (more complete) source as the seed and feeding the points to the rest for data extraction.

Multi-Source Seed-Driven (MSSD) is a function that takes as input a set of sources $S_1, S_2 \ldots S_k$ and outputs their corresponding
sets of locations \{L_{S_1}, L_{S_2}, ..., L_{S_k}\} obtained from the seed-driven approach in Alg. 1. For example, let us suppose that the seed provides a location with coordinates (57.05, 9.92) as in Fig. 1. We can search for locations across sources within the circle with center (57.05, 9.92) and a predefined radius. The different colors in the figure represent the different sources. We can discover three locations from the red source, two from the blue source and two from the green source. The algorithm for the seed-driven approach is presented in Alg. 1. Selecting a good seed is important; thus, we start by getting the most complete source (with the most points) in line 4. The points in the seed indicate regions of interest and are used for the API request in the sources. So, for each point in seed (for each \( p \) in \( P \)), we query the rest of the sources except the seed source. Line 7 shows the general API call for each of the sources, which is performed in correspondence to the requirements of the source. The request takes the coordinates of \( p \) and the radius \( r \). The search returns a set of locations \( L_p \) which is unioned to our source-specific output \( L_S \).

### 5.2 Optimizing the Radius

We can improve further \( L_{SSD} \) by adapting the API request to the source. Even though a big radius might seem like a better solution, note that the API retrieves only a fixed size sample of the underlying data. Hence, if we query with points that are nearby, we might retrieve intersecting samples. We denote the maximal result size of the API for source \( S \) as \( M_S \). Let us consider the example in Fig.3, where \( M_S \) is 3, which means that the API can not retrieve more than 3 tweets. If we query with a big radius as in the left part of Fig.3, we might get 2 out of 3 tweets in the intersection. If the radius is small, then we explore better the dense areas, but we might miss in sparser ones like in the right part of Fig. 3. The union of tweets in both searches is just 4, where ideally it should have been 6. We propose two improvements: using the knowledge of the seed to define the radius and recursively learn a suitable radius for the source.

#### Multi-Source Seed-Driven Recursive \( L_{SSD-R} \)

The advantage of \( L_{SSD-D} \) and \( L_{SSD-N} \) is that no API call is needed to adjust the radius because these calculations are performed on the seed points. However, there is a need for a better approach to assigning a suitable radius for a specific area. We propose a solution that adjusts the radius while querying the source. If an area not dense, we can identify it from the API call. However, in contrast to SQL queries on a database where the operations are transparent, the operations of API queries on an online source are a black box, and thus, we need to assume a certain level of transparency. Therefore, we assume that if the area contains less than \( M_S \) locations, the API call will retrieve all of them.

---

**Algorithm 1 Multi-Source Seed-Driven (MSSD)**

**Input:** A set of sources \{\( S_1, S_2, ..., S_k \)\}, radius \( r \)

**Output:** \( \{L_{S_1}, L_{S_2}, ..., L_{S_k}\} \)

1: for each \( S \) in \( \{S_1, S_2, ..., S_k\} \) do
2: \( L_S \leftarrow L_1 \) /* Initialize \( L_S \) of each source with \( L_1 \)*/
3: end for
4: for each \( S \) in \( \{S_1, S_2, ..., S_k\} \) do
5: end for
6: for each \( p \) in \( P \) do
7: \( L_p \leftarrow \text{API}(p, r) \) /* API request for the source \( S \)*/
8: \( L_S \leftarrow L_S \cup L_p \)
9: end for
10: end for

return \( \{L_{S_1}, L_{S_2}, ..., L_{S_k}\} \)

---

**Algorithm 1a** MSSD Density-Based (MSSD-D)

5.a: Find \( N = \{q | q \in \text{Circle}(p, r)\} \) /* Find how dense the region is*/
5.b: \( r_d = \frac{|N|}{d} \) /* Adjust the radius*/

**Algorithm 1b** MSSD Nearest Neighbor (MSSD-N)

5.a: Find \( q = \min_{q \in L_p} |p - q| \) /* Find the nearest neighbor*/
5.b: \( r_n = |p - q| \) /* Adjust the radius*/

---

**Table 1: Summary of limitations of social networks**

<table>
<thead>
<tr>
<th>API limitations</th>
<th>Krak</th>
<th>Yelp</th>
<th>Google Places</th>
<th>Foursquare</th>
<th>Twitter</th>
<th>Flickr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>10KB/month</td>
<td>3K/day</td>
<td>1/day (from 6/2018)</td>
<td>180/15 min</td>
<td>34KB/hour</td>
<td></td>
</tr>
<tr>
<td>Max Res. Size</td>
<td>100</td>
<td>50</td>
<td>20</td>
<td>50</td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>Hit Access</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Full</td>
<td>2 weeks</td>
<td>Full</td>
</tr>
<tr>
<td>supp Results</td>
<td>4.3%</td>
<td>17.3%</td>
<td>0.3%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Complete access</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Cost</td>
<td>not stated</td>
<td>negotiable</td>
<td>from 205K/month</td>
<td>from 39K/month</td>
<td>140K - 249K/month</td>
<td>not stated</td>
</tr>
</tbody>
</table>

**Table 2: Locations and points per source**

<table>
<thead>
<tr>
<th>Category</th>
<th>Krak</th>
<th>Yelp</th>
<th>GP</th>
<th>FSQ</th>
<th>TW</th>
<th>Flickr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locations</td>
<td>143,073</td>
<td>473</td>
<td>380</td>
<td>1,097</td>
<td>115</td>
<td>4,084</td>
</tr>
<tr>
<td>Points</td>
<td>32,461</td>
<td>467</td>
<td>356</td>
<td>1,093</td>
<td>25</td>
<td>2,272</td>
</tr>
</tbody>
</table>

---

**Algorithm 1a** MSSD Density-Based (MSSD-D)

5.a: Find \( N = \{q | q \in \text{Circle}(p, r)\} \) /* Find how dense the region is*/
5.b: \( r_d = \frac{|N|}{d} \) /* Adjust the radius*/

**Algorithm 1b** MSSD Nearest Neighbor (MSSD-N)

5.a: Find \( q = \min_{q \in L_p} |p - q| \) /* Find the nearest neighbor*/
5.b: \( r_n = |p - q| \) /* Adjust the radius*/

---

**Multi-Source Seed-Driven Density-Based** \( L_{SSD-D} \). The radius in this version is defined by the density of points in the seed. Before the API requests, we check the density of points in the search area in the seed, and we adapt the radius accordingly. Fig. 2a illustrates the intuition behind \( L_{SSD-D} \). We are using the point in red as seed point \( p \). Before performing any API call, we check how many points of the seed are in the search area (\( N = 4 \) points in the black circle with center \( p \) and initial radius \( r \)). Second, we adjust the radius according to the density, so in this case, we divide the radius by \( 4 (r_d = \frac{r}{4}) \). Finally, we perform the API call to the source with the red circle. Alg. 1a shows the alterations we make in Alg. 1 for the radius calculation. We add line 5.a and 5.b after line 5 in Alg. 1. First, we find the density of the region, and then, we adjust the radius depending on the density. We query with the adjusted \( r_d = \frac{r}{N} \) in line 7 of Alg. 1.

---

**Multi-Source Seed-Driven Nearest Neighbor** \( L_{SSD-N} \). As the name suggests, we use the nearest neighbor in the seed to define the radius. A simple illustration of this idea is presented in Fig. 2b. For each of the points \( p \) in the seed (in red), we find the nearest neighbor \( q \) in the seed (in green), and then we query with the adjusted radius \( r_n = |p - q| \). Note that we query with a small radius in dense areas and a big radius in sparse areas. Alg. 1b instead adds line 5.a and 5.b after line 5 of Alg. 1. It finds the nearest neighbor \( q \) of the point \( p \). Then, we set \( r_n = |p - q| \). The adjusted \( r_n \) is used to query the sources in line 7 of Alg. 1.

---

**Multi-Source Seed-Driven Recursive** \( L_{SSD-R} \). The advantage of \( L_{SSD-D} \) and \( L_{SSD-N} \) is that no API call is needed to adjust the radius because these calculations are performed on the seed points. However, there is a need for a better approach to assigning a suitable radius for a specific area. We propose a solution that adjusts the radius while querying the source. If an area not dense, we can identify it from the API call. However, in contrast to SQL queries on a database where the operations are transparent, the operations of API queries on an online source are a black box, and thus, we need to assume a certain level of transparency. Therefore, we assume that if the area contains less than \( M_S \) locations, the API call will retrieve all of them.
The API retrieves a sample of size $M_S$ of the underlying data in a queried region $\text{Circle}(p, r)$. If the underlying locations $L_p' (S)$ are already less than $M_S$, then we assume that the API will retrieve all the locations lying in $\text{Circle}(p, r)$. For example, if there are 30 locations in $\text{Circle}(56.78 \text{ km}, 1)\text{ km}$ and $M_S = 50$, then querying with $p = (56.78 \text{ km})$ and $r = 1 \text{ km}$ will return all 30 locations.

**Theorem 1.** Let $p, r > 0$ be a pair of point and radius such that $\text{API}(p, r) = L_p'$ where $|L_p'| < M_S$. Then, for all $r'$ such that $r' < r$, $L_p'' \subseteq L_p'$. 

**Proof.** Let us assume that there are $|L_p'(S)|$ locations in $\text{Circle}(p, r)$ and $|L_p''(S)|$ locations in $\text{Circle}(p, r')$. Since $r' < r$, then the surface covered by $\text{Circle}(p, r')$ is smaller than the surface covered by $\text{Circle}(p, r)$ ($\pi r'^2 < \pi r^2$). Consequently, $L_p'(S) \subseteq L_p''(S)$. According to Assumption 1, since $\text{API}(p, r)$ retrieves $|L_p'| < M_S$, then $L_p' = L_p(S)$ and $|L_p'(S)| < M_S$. Given that $L_p''(S) \subseteq L_p'(S)$ and $|L_p'(S)| < M_S$, we conclude that $|L_p''| < M_S$ and $L_p'' = L_p'(S)$. Finally, from $L_p''(S) \subseteq L_p'(S)$, we derive that $L_p'' \subseteq L_p'$.

This is an important finding that will be used in defining MSSD-R. Since there are less than $M_S$ locations retrieved by the API call in source $S$, there are no new locations to be gained by querying with a smaller radius. Thus, we propose a recursive method that uses Theorem 1 as our stopping condition. First, we query with an initial large radius, and if the result size is $M_S$, then we know this is a dense area, and we perform another request with a smaller radius. The search stops when the number of returned results is smaller than the maximal result size because according to Theorem 1. The recursive method is illustrated in Fig. 2c. Let us suppose that we are querying source $S$ and the maximal result size is $M_S = 5$. After querying with the green circle, we get 5 locations so we know that the area is dense. We reduce the radius by $\alpha (r_p = \frac{r}{2})$, and now try again with the blue circle. We get 5 locations again, therefore we continue once more with a smaller radius. When we query with the red circle, we get only 2 locations, so we stop. Alg. 1c (MSSD-R) has a modification where a new algorithm is called. Instead of querying with a static $r$, we perform a recursive procedure to adjust the radius. Alg. 2 takes the following parameters: the radius $r_p$, the coefficient $\alpha$ which is used to reduce the radius, the point $p$ that comes from the seed, the queried source $S$ and the locations $L_p'$. The stopping condition is retrieving less than $M_S$ locations (line 3). However, we have no control over the number of additional requests needed by MSSD-R but under the following assumption, we know that MSSD-R converges:

**Assumption 2.** Let $S$ be a source and $L(S)$ its locations. For each point $p$ there exists a radius $r_p$ such that the surface covered by $\text{Circle}(p, r_p)$ contains less than $M_S$ locations.

We assume that there will always be an $r_p$ such that $|L_p'| < M_S$. MSSD-R performs several requests decreasing $r$ by $\alpha$ until $r_p$ is found and MSSD-R reaches the stopping condition. Given Assumption 2, we guarantee that MSSD-R performs a finite number of requests.

**Algorithm 1c MSSD Recursive (MSSD-R)**

```plaintext
6: for each $S$ in $(S_1, S_2, \ldots, S_k)$ · $S_{seed}$ do
7: $L_p' \leftarrow \emptyset$
8: $L_p' \leftarrow \text{RadRecursive}(r, \alpha, p, S, L_p')$
9: $L_S \leftarrow L_S \cup L_p'$
10: end for
```

**Algorithm 2 RadRecursive**

Input: $r_p, \alpha, p, S, L_p'$
Output: $L_p''$

```plaintext
1: $R \leftarrow \text{API}(p, r_p, S)$ →/* Query $S$ with $r_p$*/
2: $L_p'' \leftarrow L_p' \cup R$
3: if $|R| < M_S$ then
4: return $L_p''$ →/* The area is not dense*/
5: else
6: $\text{RadRecursive}(\alpha r, \alpha, p, S, L_p')$ →/* Call with new $r_p$*/
7: end if
```

### 5.3 Optimizing the Point Selection

All MSSD algorithms are based on exhaustive querying. However, some seed points might be quite close to each other, resulting in redundant API requests. We propose MSSD’ which clusters the
seed points using DBSCAN [8] (a clustering algorithm suitable for spatial data and robust to noise) and queries with the centroids of the clusters. If the results size is maximal, then there is a possibility that this is a dense area. Afterward, we apply DBSCAN on the union of the points of the current cluster and the points retrieved from the API request. Depending on the data distribution of the source, we move the focus to the dense areas that we discover. We stop when the result size of the request is less than the maximal (based on Theorem 1).

Fig. 4 shows a simple example of MSSD*. The seed points come from the seed, whereas the blue ones are in the source. The initial DBSCAN will cluster together (A, B, C), (D, E), (F, G, H) and I. After the querying with the centroids of these clusters, only clusters (A, B, C) and I will continue further. The new clusters for (A, B, C) will be A, B and (C, K, L), where K and L are points from the source. For cluster I, we query with the centroid of (I, J, M). In the third step cluster (I, J, M) is divided to I, (J,M) and N, where N is a new point discovered from the second step. MSSD* is formalized in Alg. 3. After a source is chosen, its points are clustered with DBSCAN (line 5) using \( e \) as minimal distance between points and \( m \) as the number of points that a cluster should have. DBSCAN returns the set of clusters \( C \). For each centroid \( c \) of the cluster \( C \), we call RadRecursive\(^*\) (Alg. 2a), which is similar to its parent version, RadRecursive (Alg. 2) regarding the stopping condition and the adaptive radius but differs from line 6 and on (the else clause). If the area is dense, then we split the cluster by taking into consideration the union \( C_p \cup R \) of points in the cluster \( C_p \) and \( R \) with DBSCAN in line 6 and we receive a new set of clusters \( \{C^r\} \). For each centroid \( c^r \) of the cluster \( C^r \) we call the algorithm again with the adjusted parameters. Note that in the case of Twitter, the majority of results \( R \) is polygons. Therefore we modify line 6 in Alg. 2a with (i) the centroids of the polygons and we denote this version of MSSD*-C or (ii) the nearest point of the polygon to the queried point \( p \), and we denote this version as MSSD*-N.

**Algorithm 2a RadRecursive\(^*\)**

| Input: | \( r, \alpha, \beta, p, C_p, S, e, m, L_p^r \) |
| Output: | \( L_p^r \) |
| 6: | \( \{C^r\} \leftarrow \text{DBSCAN}(C_p \cup R, \frac{e}{2}, \frac{e}{m}) \) → /* DBSCAN on the union of \( C_p \) and \( R \) with new parameters\(^*\) |
| 7: | for each \( c^r, C^r > \) do |
| 8: | RadRecursive\((\frac{e}{2}, \alpha, \beta, C^r, S, \frac{e}{2}, \frac{e}{m}, L_p^r)\) |
| 9: | end for |

MSSD\(^*\) has these advantages: (i) MSSD\(^*\) manages better the requests by using the centroids of clusters rather than all the points in a cluster; (ii) MSSD\(^*\) is not sensitive to parameters because it uses an adaptive algorithm to learn them for each of the sources, and (iii) while querying, MSSD\(^*\) adapts the center of the circle depending on the locations found by the previous query. Let us now suppose that the optimal combination of pairs of \( p^*, r^* > \) that retrieve the maximal \( L^* \) exists. In order to compare our solution to the optimal, let us first prove the submodularity of our problem.

**Theorem 2.** Optimizing the data extraction based on API calls is a monotone submodular problem.

<table>
<thead>
<tr>
<th>Algorithm 3 MSSD(^*) algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>1:</td>
</tr>
<tr>
<td>2:</td>
</tr>
<tr>
<td>3:</td>
</tr>
<tr>
<td>4:</td>
</tr>
<tr>
<td>5:</td>
</tr>
<tr>
<td>6:</td>
</tr>
<tr>
<td>7:</td>
</tr>
<tr>
<td>8:</td>
</tr>
<tr>
<td>9:</td>
</tr>
<tr>
<td>10:</td>
</tr>
<tr>
<td>11:</td>
</tr>
<tr>
<td>12:</td>
</tr>
<tr>
<td>13:</td>
</tr>
</tbody>
</table>

**Proof.** An API call takes \( < p, r > \) as parameters and retrieves \( L_p^r \) locations. Let us denote as \( \gamma(p, r) \) the gain (new locations) that \( API(p, r) \) brings. Note that an extra API call is effective as long as it contributes to the union of the results of the previous calls. To prove the submodularity, we need to show that \( \gamma(p^* \cup p, r) \geq \gamma(p \cup p, r) \) if \( P^* \subset P \). Let us consider a set of points \( P \) and \( P^* \) such that \( P^* \subset P \). The locations retrieved from \( P \) are \( \bigcup_{i=1}^{P} L_p^r \) and the locations retrieved from \( P^* \) are \( \bigcup_{i=1}^{P^*} L_p^r \). Since \( P^* \subset P \), \( \bigcup_{i=1}^{P^*} L_p^r \subseteq \bigcup_{i=1}^{P} L_p^r \). Let us consider a new point \( p \) and \( L^r_p \) the result of \( API(p, r) \). Since \( \bigcup_{i=1}^{P^*} L_p^r \subseteq \bigcup_{i=1}^{P} L_p^r \), then \( (L_p^r \cap \bigcup_{i=1}^{P^*} L_p^r) \subseteq (L_p^r \cap \bigcup_{i=1}^{P} L_p^r) \). As a result, \( \gamma(p^* \cup p, r) \geq \gamma(p \cup p, r) \). To prove the monotonicity, for every \( \epsilon \), \( \gamma(p \cup \epsilon, r) \geq \gamma(p, r) \). To prove the monotonicity, for every \( \epsilon \), \( \gamma(p \cup \epsilon, r) \geq \gamma(p, r) \). To prove the monotonicity, for every \( \epsilon \), \( \gamma(p \cup \epsilon, r) \geq \gamma(p, r) \). To prove the monotonicity, for every \( \epsilon \), \( \gamma(p \cup \epsilon, r) \geq \gamma(p, r) \). To prove the monotonicity, for every \( \epsilon \), \( \gamma(p \cup \epsilon, r) \geq \gamma(p, r) \). Hence, \( \bigcup_{i=1}^{P^*} L_p^r \subseteq \bigcup_{i=1}^{P} L_p^r \). □

Our MSSD\(^*\) tries to solve the data extraction problem by providing a solution that starts with initial centroids and then splits further if the area looks promising in terms of density. However, we extract only \( M_S \) locations in one call, and this sample might not be representative if the amount of the actual locations in the area may be quite large. So, if the sample of the \( M_S \) points misses some dense areas, our DBSCAN will classify these as outliers, and we will not query further. Thus, our solution is greedy because it makes a locally optimal solution regarding which API calls to perform in step \( i+1 \) based on the information of step \( i \).

**Theorem 3.** The greedy solution MSSD\(^*\) of our monotone submodular problem performs at least \( 1 - \frac{1}{e} \) as good as the optimal solution [26]. The proof uses the submodularity and the monotonicity to show the ratio between the greedy and the optimal solution.

**Proof.** Let \( L^* \) be the result of the optimal solution from points \( P^* \) and \( L_k^* \) the greedy solution provided by MSSD\(^*\) for \( n \) requests. Note that \( L^* \) is not the same as the total locations \( L(S) \) of the source.
We run MSSD with a nearest-neighbor method to define a flexible radius (Alg. 3) to with small values of $\alpha$ are rewarded with a higher percentage of locations. For example, for 0.3% more requests, we retrieve 18.8% more locations in Foursquare. In Flickr, for 0.8% more requests, we retrieve 17.5% more locations. Similarly, starting with a big radius is safer and more rewarding. For instance, Foursquare and Yelp perform less than 0.4% of the requests to get around 46% more locations when starting with $r = 16$ km compared to $r = 1$ km. A risk-averted selection of parameters turns out to provide a good trade-off between the number of requests and number of locations because MSSD* adapts to the density of the region and still manages the requests carefully. Thus, the algorithm is robust to different parameter settings, and fine-tuning is not needed.

Choosing a different seed. In order to show that our MSSD algorithms apply to any type of seed (preferable a rich source), we ran MSSD-D, MSSD-N, MSSD-R and MSSD* using as seed Flickr, Foursquare, Yelp, Google Places, and Twitter. The results are presented in Fig. 7. Even though Krak performs the best, the other sources which provide significantly fewer seed points (see Table 2) are able to achieve comparable results. Recall that Krak has 14 times more seed points than Flickr, 50 times more than Foursquare, 70 times more than Yelp, 91 times more than Google Places and 295 times more than Twitter. Apart from Krak, MSSD* performs the best for Flickr, Yelp, and Foursquare. For Flickr, MSSD* with Yelp seed points retrieves 6.5 times more points than SI. For FSQ, MSSD* with Flickr and Yelp seed points retrieves 2.9 and 2.5 times more points than SI, respectively. For Yelp, MSSD* with Flickr seed points retrieves 3.5 times more data than SI, whereas Krak retrieves
Figure 5: Requests versus locations for different MSSD algorithms with Krak as seed

Table 3: % of req. vs % of loc. for MSSD* relative to MSSD-R depending on α and r

<table>
<thead>
<tr>
<th>Sources</th>
<th>Req</th>
<th>Alpha</th>
<th>Radius</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>FSQ</td>
<td>13.2%</td>
<td>13.0%</td>
<td>12.9%</td>
</tr>
<tr>
<td></td>
<td>% of loc</td>
<td>% req</td>
<td>% of loc</td>
</tr>
<tr>
<td>Flickr</td>
<td>15.0%</td>
<td>14.9%</td>
<td>14.8%</td>
</tr>
<tr>
<td></td>
<td>% of loc</td>
<td>% req</td>
<td>% of loc</td>
</tr>
<tr>
<td>GP</td>
<td>9.3%</td>
<td>9.1%</td>
<td>8.8%</td>
</tr>
<tr>
<td></td>
<td>% of loc</td>
<td>% req</td>
<td>% of loc</td>
</tr>
<tr>
<td>Yelp</td>
<td>17.5%</td>
<td>17.2%</td>
<td>16.8%</td>
</tr>
<tr>
<td>Twitter-C</td>
<td>5.0%</td>
<td>5.0%</td>
<td>5.0%</td>
</tr>
<tr>
<td>Twitter-N</td>
<td>99.5%</td>
<td>99.4%</td>
<td>99.0%</td>
</tr>
</tbody>
</table>

Figure 6: Nr of loc. per source

Figure 7: MSSD results with different seeds for all sources

3.6 times more while having 70 times more seed points. For Twitter and Google Places, MSSD-R performs the best; 7.4 times more locations than SI with Flickr seed points in Twitter and 23.6 times more locations with Yelp seed points in Google Places. The performance of each algorithm in terms of the number of requests versus the number of locations can be found in [14]. An interesting observation is that MSSD* sometimes performs better than MSSD-R even when the seed source is not rich, MSSD* manages to achieve good results due to its ability to adapt the next call according to the distribution of the source.

Elapsed time of the experiments. The elapsed time is more important than the CPU time because of the bandwidth limitations. Our experiments were run simultaneously for all the sources in order to avoid the temporal bias, so all bandwidth limitations were
respected at the same time. The elapsed time for all the sources is around 1 week for the MSSD-F, MSSD-D and MSSD-N, 2 weeks for MSSD-R and 1.7 days for MSSD*. If the algorithms are run independently, it takes on average 1 day per source for MSSD-F, MSSD-D and MSSD-N, 2 days for MSSD-R and less than 6 hours for MSSD*.

6.2 Comparison with Existing Baselines

The technique using linked accounts [1, 13, 28] requires users that have declared their account in another social network. From our initial querying of the sources, there were only 0.27 % of users on Flickr with linked accounts to Twitter and 0.003 % of users on Twitter with linked accounts to Foursquare. Hence, a comparison with this technique makes little sense. The keyword-based querying shows limited applicability in location-based data retrieval. We conducted a small experiment using the names of cities and towns in North Denmark as keywords. For Flicker and Twitter, the precision (% of data that falls in the queried region) was just 31.6% and 0.85% respectively, while the recall was less than 5%, relative to MSSD*. Foursquare and Yelp offer a query by term or query by location expressed as a string. The former [query by term] does not retrieve any data when queried with a city or town name. If we express the location as a string, the precision is 93% and 85% for Foursquare and Yelp respectively, and the recall is less than 19%, relative to MSSD*. In GP, the data retrieved is only the towns and the cities themselves. For example, if we query with the keyword "Aalborg", the API will return Aalborg city only and not any other places located in Aalborg (1 request per 1 location). Even though the precision is 100%, the recall is only 0.07% relative to MSSD*. Thus, we compare to Snowball and to the technique mentioned in [19].

To compare with Snowball ([12, 31]), we formed the seed with the users found in Section 4. We used the same number of requests for Snowball and MSSD*. Snowball is based on users, and consequently, it can be applied only to Twitter and Flickr (Foursquare API no longer provides the check-in data unless the crawling user has checked in himself at the venue). The technique mentioned in [19] (we will refer to it as Self-seed) starts with querying a specific location to get initial points. Later, other requests are performed using the seed points of the previous step. We ran Self-seed on all our sources for the same number of requests as MSSD* (results in Fig. 8). Snowball in Twitter retrieves more locations in the region than versions of MSSD* (MSSD*-C and MSSD*-N) and Self-seed (Fig. 8a) because in the case of user-based calls, the bandwidth is 200 (100 for location-based) and the historical access is unlimited (2 weeks for location-based). In the case of Flickr, MSSD* outperforms Snowball and Self-seed with 14 and 3 times more locations respectively. Snowball gets most of the data in the region in the beginning and degrades later because when using Snowball, while we traverse the network, there is more and more data which falls outside the region of interest. MSSD* yields a higher number of locations compared to Self-seed: 5.5 times more locations for Foursquare, 9 times more locations for Yelp and 3.5 times more locations for Google Places.

Self-seed in the case of directories stops yielding new locations after approximately 500 requests. In the case of directories, after some steps, the seed points in Self-seed stop growing, converging into a dead end. Recall that Google Places has a result size of 20 and is denser in terms of data, so it has new locations for the subsequent steps, avoiding thus the convergence. The number of users and the time period covered are presented in Table 5. Despite the slight advantage of Snowball on Twitter in terms of the number of locations, the data comes only from 35 users compared to 231 for MSSD*-N and 101 for Self-seed. Moreover, the time period covered by the tweets in Snowball is 3 years compared to 1 year of MSSD versions. Regarding Flicker, the time period of the photos is the same, but the number of photos and the number of users are 1-2 orders of magnitude larger for MSSD* compared to Snowball. Self-seed can retrieve a better variety of users and locations compared to Snowball but still contains only half the number of locations and users of MSSD*.

### Table 4: MSSD-R and MSSD* compared to ground truth

<table>
<thead>
<tr>
<th>Source</th>
<th>Algorithm</th>
<th>Real 80%</th>
<th>Synthetic 20%</th>
<th>Req</th>
<th>Real 70%</th>
<th>Synthetic 30%</th>
<th>Req</th>
<th>Real 60%</th>
<th>Synthetic 40%</th>
<th>Req</th>
<th>Real 50%</th>
<th>Synthetic 50%</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>MSSD-R</td>
<td>85.62%</td>
<td>87.16%</td>
<td>10.16%</td>
<td>84.44%</td>
<td>85.70%</td>
<td>10.12%</td>
<td>82.71%</td>
<td>83.70%</td>
<td>9.80%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yelp</td>
<td>MSSD-R</td>
<td>97.17%</td>
<td>99.44%</td>
<td>12.74%</td>
<td>97.10%</td>
<td>99.4%</td>
<td>12.49%</td>
<td>97.10%</td>
<td>99.23%</td>
<td>11.78%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FSQ</td>
<td>MSSD-R</td>
<td>94.27%</td>
<td>96.69%</td>
<td>10.67%</td>
<td>94.19%</td>
<td>95.51%</td>
<td>10.62%</td>
<td>94.07%</td>
<td>95.26%</td>
<td>10.45%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GP</td>
<td>MSSD-R</td>
<td>92.42%</td>
<td>78.76%</td>
<td>13.85%</td>
<td>91.60%</td>
<td>76.15%</td>
<td>13.19%</td>
<td>89.94%</td>
<td>69.33%</td>
<td>13.49%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>MSSD-R</td>
<td>84.16%</td>
<td>97.70%</td>
<td>2.45%</td>
<td>81.16%</td>
<td>95.66%</td>
<td>2.44%</td>
<td>80.99%</td>
<td>65.99%</td>
<td>2.45%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 5: Snowball, Self-seed and MSSD* comparison

<table>
<thead>
<tr>
<th>Source</th>
<th>Nr of loc.</th>
<th>Nr of users</th>
<th>Time period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter</td>
<td>Snowball</td>
<td>1421</td>
<td>35</td>
</tr>
<tr>
<td>Self-seed</td>
<td>851</td>
<td>101</td>
<td>2017-2018</td>
</tr>
<tr>
<td>MSSD*-C</td>
<td>936</td>
<td>195</td>
<td>2017-2018</td>
</tr>
<tr>
<td>MSSD-N</td>
<td>1237</td>
<td>231</td>
<td>2017-2018</td>
</tr>
<tr>
<td>MSSD*</td>
<td>39427</td>
<td>1740</td>
<td>2005-2018</td>
</tr>
</tbody>
</table>

6.3 MSSD-R and MSSD* Result Completeness

Given the API limitations, we cannot get the actual ground truth of source locations. Instead, we performed the following experiment: first, we union all the locations sets from all our algorithms (SI, MSSD-F, MSSD-D, MSSD-N, MSSD-R and MSSD*) to create a dataset of real data; second, we learn the distribution \( D \) of the locations by dividing the area in a grid of 1km x 1km and assigning each grid cell a probability \( p_d \sim D_d \); third, we generate synthetic locations in the area and assign them to a grid cell \( d \) with the estimated probability \( p_d \). We consider the synthetic and the real data as ground truth. We implemented "simulated offline" API functions for each source, respecting the maximal result size for each of them. We ran our MSSD-R and MSSD* on the ground truth data for different ratios of synthetic data as in Table 4. The data retrieved by MSSD-R is above 94% of the ground truth in Yelp and Foursquare and above 80% of the ground truth in Flickr, Google Places, and Twitter. MSSD* performs the best in Yelp and Twitter (MSSD*-N) with above 70% of the ground truth for all ratios of real versus synthetic data, followed...
This paper was motivated by the need for an efficient algorithm that uses one source as the seed. Foursquare, Twitter, and Flickr. Then, we proposed a seed-driven algorithm that finds the API limitations for six sources: Krak, Yelp, Google Places, Foursquare, Twitter, and Flickr, the activity originates from a larger base of users (up to 6.6 times more), and (iii) in the case of directories, our MSSD avoids converging into a dead end. In a ground truth dataset, for most of the sources, our MSSD-R algorithm finds 82% - 99% of the ground truth, while MSSD* with 10% of the requests is able to guarantee 63% - 73% of the ground truth.

6.4 Discussion of Experiments
Selecting an external seed of points improved the number of locations retrieved and avoid converging into a dead end like in Self-seed. Moreover, the attempts to adapt the radius of the search according to the search region prove to be effective in retrieving more locations. MSSD-F, MSSD-D and MSSD-N extract on average up to 11.1 times more data than SI but if we adapt the radius according to the source (MSSD-R), we extract up to 14.3 times more locations than SI. MSSD* provides a very good trade-off between the number of requests and number of locations as MSSD* extracts up to 90% of the data of MSSD-R with less than 16% of its requests. Our comparison with the Snowball and the Self-seed baseline shows that our seed-driven algorithm is better in terms of extracting (i) up to 14 times more locations for all the sources, (ii) in the case of Twitter and Flickr, the activity originates from a larger base of users (up to 6.6 times more), and (iii) in the case of directories, our MSSD avoids converging into a dead end. In a ground truth dataset, for most of the sources, our MSSD-R algorithm finds 82% - 99% of the ground truth, while MSSD* with 10% of the requests is able to guarantee 63% - 73% of the ground truth.

7 CONCLUSIONS AND FUTURE WORK
This paper was motivated by the need for an efficient algorithm that extracts recent geo-social data. We formulated the problem of data extraction as an optimization problem which aims to maximize the retrieved locations while minimizing the requests. We identified the API limitations for six sources: Krak, Yelp, Google Places, Foursquare, Twitter, and Flickr. Then, we proposed a seed-driven algorithm that uses one source as the seed to feed the points as API parameters to the others. MSSD versions extracted up to 14.3 times more data than SI. Our optimized algorithm MSSD* retrieved 90% of the locations with less than 16% of the requests, outperforming MSSD-D and MSSD-N. Interesting directions for future research include applying machine learning for data extraction, seed selection based on other criteria (diversity in semantics, maximal spread of points, relation to the source), data integration, and data fusion of location-based data from multiple geo-social sources.

REFERENCES
[18] Anjika L, Srikanta B, and Deepak P. 2017. LoCat: Influence Quantification for Location Promotion in Location-Based Social Networks. In ICATI.
[34] M. Weiler, K. A. Schmid, N. Mamoulis, and M. Renz. 2015. Geo-social co-location mining. In GeoRips@SIGMOD.
[35] Z. Yao, Y. Fu, B. Liu, Y. Liu, and H. Xiong. 2016. POI recommendation: A temporal matching between POI popularity and user regularity. In ICDM.
[38] J. Zhang and C. Chow. 2013. iGSLR: personalized geo-social location recommendation on location-based social networks. In KDD.