Seed-Driven Geo-Social Data Extraction

Isaj, Suela; Pedersen, Torben Bach

Published in:
Proceedings of the 16th International Symposium on Spatial and Temporal Databases, SSTD 2019

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

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

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 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

SSTD '19, August 19–21, 2019, Vienna, Austria

ACM ISBN 978-1-4503-6280-1/19/08…$15.00
https://doi.org/10.1145/3340964.3340973

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, 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 (sparsely 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 user. 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 $API: \mathcal{P} x \mathcal{R}^+ \Rightarrow 2^{L(S)}$, where $\mathcal{P}$ is the domain of geographical points and $\mathcal{R}^+$ is the domain of non-negative numbers. $API(p, r)$ queries with a centroid $p \in \mathcal{P}$ and a radius $r \in \mathcal{R}$ 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_S of the underlying activities $L_p^r(S)$ in $Circle(p, r)$. So, if $M_S = 100$, 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 $\{l_1, l_4, l_5, l_6\}$ and the second request retrieves $\{l_2, l_4, l_5, l_6\}$, then the second request contributed with only one new location (l_2).

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 $L_i = \bigcup_{j=1}^{n} L_{p_j}^{r_j}$ 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

API limitations	Krak	Yelp	Google Places	Foursquare	Twitter	Flickr
Bandwidth	10K/month	5K/day	1/day (from 6/2018)	550/hour	180/15 min	3.6K/hour
Max Res. Size	100	50	20	50	100	500
Hist. Access	N/A	N/A	N/A	Full	2 weeks	Full
Supp Results	4.3%	17.3%	0.5%	0.0%	0.0%	0.0%
Complete access	yes	yes	yes	yes	1%	yes
Cost	not stated	negotiable	from 200$/month	from 599$/month	149$ - 2499$/month	not stated

Table 1: Summary of limitations of social networks

Category	Krak	Yelp	GP	FSQ	TW	Flickr
Locations	143,073	473	380	1,097	115	4,084
Points	32,461	467	356	1,093	25	2,272

Table 2: Locations and points per source

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^r, which is unioned to our source-specific output L_S.

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

Input: A set of sources $\{S_1, S_2, \ldots S_k\}$, radius r
Output: $\{L_{S_1}, L_{S_2}, \ldots L_{S_k}\}$
1: **for each** S in $\{S_1, S_2, \ldots S_k\}$ **do**
2: $L_S \leftarrow L_I$ →/* Initialize L_S of each source with L_I*/
3: **end for**
4: Let S_{seed} be the source with the most points in $\{S_1, S_2, \ldots S_k\}$, L_{seed} its locations and P the distinct points in L_{seed}
5: **for each** p in P **do**
6: **for each** S in $\{S_1, S_2, \ldots S_k\}$ - S_{seed} **do**
7: $L_p^r \leftarrow API(p, r)$ →/* API request for the source S */
8: $L_S \leftarrow L_S \cup L_p^r$
9: **end for**
10: **end for**
return $\{L_{S_1}, L_{S_2}, \ldots L_{S_k}\}$

5.2 Optimizing the Radius

We can improve further *MSSD* 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 Density-Based *MSSD-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 *MSSD-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 ($r_d = \frac{r}{|N|}$). 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.

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

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

Multi-Source Seed-Driven Nearest Neighbor *MSSD-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 ones. 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.

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

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

Multi-Source Seed-Driven Recursive *MSSD-R.* The advantage of *MSSD-D* and *MSSD-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.

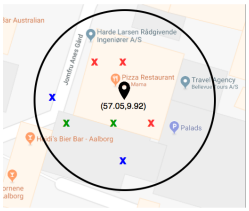
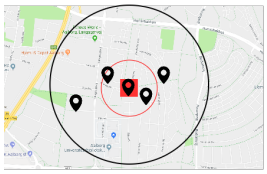
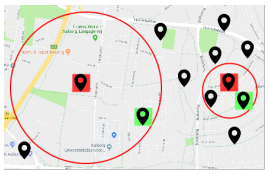


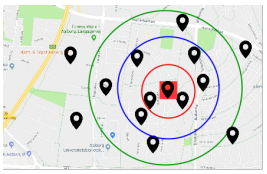
Figure 1: *MSSD* approach



(a) *MSSD-D* radius



(b) *MSSD-N* radius



(c) *MSSD-R* radius

Figure 2: *MSSD* radius

Assumption 1. *For each source S in* $\{S_1, S_2, ..., S_k\}$, *if* $Circle(p, r)$ *contains* $L_p^r(S)$ *locations such that* $|L_p^r(S)| \leq M_S$, *then* $API(p, r)$ *will retrieve* $L_p^r = L_p^r(S)$.

The API retrieves a sample of size M_S of the underlying data in a queried region $Circle(p, r)$. If the underlying locations $L_p^r(S)$ are already less than M_S, then we assume that the API will retrieve *all* the locations lying in $Circle(p, r)$. For example, if there are 30 locations in $Circle((56.78\ 9.67), 1km)$ and $M_S = 50$, then querying with $p = (56.78\ 9.67)$ and r=1 km will return all 30 locations.

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

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

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 α ($r_r = \frac{r}{\alpha}$) and query 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_r, the coefficient α which is used to reduce the radius, the point p that comes from the seed, the queried source S and the locations L_p^r. 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:

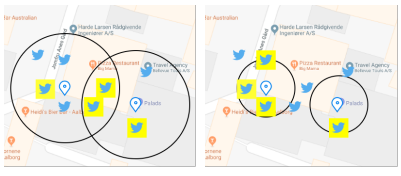


Figure 3: Radius adjustment

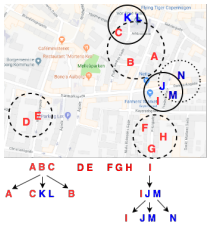


Figure 4: *MSSD

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* $Circle(p, r_p)$ *contains less than* M_S *locations.*

We assume that there will always be an r_p such that $|L_p^{r_p}| < M_S$. *MSSD-R* performs several requests decreasing r by α 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*)

```
6:  for each S in {S_1, S_2, ...S_k} - S_seed do
7:      L_p^r ← ∅
8:      L_p^r ← RadRecursive(r_r, α, p, S, L_p^r)
9:      L_S ← L_S ∪ L_p^r
10: end for
```

Algorithm 2 RadRecursive

```
Input: r_r, α, p, S, L_p^r
Output: L_p^r
1: R ← API(p, r_r, S)          →/* Query S with r_r */
2: L_p^r ← L_p^r ⋃ R
3: if |R| < M_S then
4:     return L_p^r            →/* The area is not dense*/
5: else
6:     RadRecursive(r_r/α, α, p, S, L_p^r)   →/* Call with new r_r */
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 ϵ 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 $C_p \bigcup R$ of points in the cluster C_p and the retrieved points from the source R. We cluster $C_p \bigcup R$ with DBSCAN in line 6 and we receive a new set of clusters $\{C'\}$. For each centroid c' of the cluster C' 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: $r_r, \alpha, < p, C_p >, S, \epsilon, m, L_p^r$
Output: L_p^r
6: $\{C\}' \leftarrow DBSCAN(C_p \bigcup R, \frac{\epsilon}{\alpha}, \frac{m}{\alpha})$ →/* DBSCAN on the union of C_p and R with new parameters*/
7: **for each** $< c', C' >$ **do**
8: RadRecursive*($\frac{r_r}{\alpha}, \alpha, < c', C'_c >, S, \frac{\epsilon}{\alpha}, \frac{m}{\alpha}, L_p^r$)
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.*

Algorithm 3 *MSSD** algorithm

Input: A set of sources $\{S_1, S_2, \ldots S_n\}$, radius r
Output: $\{L^*_{S_1}, L^*_{S_2}, \ldots L^*_{S_k}\}$
1: **for each** S in $\{S_1, S_2, \ldots S_k\}$ - S_{seed} **do**
2: $L_S \leftarrow L_I = \bigcup_{i=1}^{k} L_i$ →/* Initialize each L_S with L_I*/
3: **end for**
4: Let S_{seed} be the source with the most points in $\{S_1, S_2, \ldots S_k\}$, L_{seed} its locations and P the distinct points in L_{seed}
5: $\{C\} \leftarrow DBSCAN(P), \epsilon, m)$
6: **for each** S **do**
7: **for each** $< c, C >$ **do**
8: $L_p^r \leftarrow \emptyset$
9: $L_p^r \leftarrow$ RadRecursive*($r, \alpha, < c, C_c >, S, \epsilon, m, L_p^r$)
10: $L_S \leftarrow L_S \cup L_p^r$
11: **end for**
12: **end for**
return $\{L^*_{S_1}, L^*_{S_2}, \ldots L^*_{S_k}\}$

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 $\bigcup_{i=1}^{|P'|} L_{p_i}^r$ and the locations retrieved from P are $\bigcup_{i=1}^{|P|} L_{p_i}^r$. Since $P' \subset P$, $\bigcup_{i=1}^{|P'|} L_{p_i}^r \subseteq \bigcup_{i=1}^{|P|} L_{p_i}^r$. Let us consider a new point p. and L_p^r the result of $API(p, r)$. Since $\bigcup_{i=1}^{|P'|} L_{p_i}^r \subseteq \bigcup_{i=1}^{|P|} L_{p_i}^r$, then $(L_p^r \cap (\bigcup_{i=1}^{|P'|} L_{p_i}^r)) \subseteq (L_p^r \cap (\bigcup_{i=1}^{|P|} L_{p_i}^r))$. As a result, $\gamma(P' \cup p, r) \geq \gamma(P \cup p, r)$. To prove the *monotonicity*, for every $P' \subseteq P$, $|\bigcup_{i=1}^{|P'|} L_{p_i}^r| \leq |\bigcup_{i=1}^{|P|} L_{p_i}^r|$. So, the more we increase the set of seed points, the more locations we get. It is simple to show that $\bigcup_{i=1}^{|P|} L_{p_i}^r = (\bigcup_{i=1}^{|P'|} L_{p_i}^r) \cup (\bigcup_{i-i}^{|P-P'|} L_{p_i}^r)$ so $(\bigcup_{i=1}^{|P'|} L_{p_i}^r) \cup (\bigcup_{i-i}^{|P-P'|} L_{p_i}^r) \supseteq \bigcup_{i=1}^{|P'|} L_{p_i}^r$. Hence, $|\bigcup_{i=1}^{|P'|} L_{p_i}^r| \leq |\bigcup_{i=1}^{|P|}$. □

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 those 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 in terms of maximizing the number of locations, where e is the base of the natural logarithm.*

A greedy approach to a monotone submodular problem is guaranteed to be 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

S but instead the optimal solution given P^* starting seed points and obeying the limitations of the API. Due to the monotonicity, we can write: $\bigcup_{i=1}^{|P^*|} L^r_{p_i} \leq \bigcup_{i=1}^{|P^* \cup P'|} L^r_{p_i} = \bigcup_{i=1}^{|P'|} L^r_{p_i} + \sum_{j=1}^{n} \gamma(p_j, r) \leq \bigcup_{i=1}^{|P'|} L^r_{p_i} + n(\bigcup_{i=1}^{|P'+1|} L^r_{p_i} - \bigcup_{i=1}^{|P'|} L^r_{p_i})$ and $\bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|P'|} L^r_{p_i} \leq n(\bigcup_{i=1}^{|P'+1|} L^r_{p_i} - \bigcup_{i=1}^{|P'|} L^r_{p_i})$. We rearrange as: $\bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|P'|} L^r_{p_i} \leq n((\bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|P'|} L^r_{p_i}) - (\bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|P'+1|} L^r_{p_i}))$ and we use δ_i to represent $\bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|P'|} L^r_{p_i}$ so we can rewrite: $\delta_i \leq n(\delta_i - \delta_{i+1})$ and finally $\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i$. So, for every $k \leq n$ we can write $\delta_k \leq (1 - \frac{1}{n})^k \delta_0$. Note that $\delta_0 = \bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|\emptyset|} L^r_{p_i} = \bigcup_{i=1}^{|P^*|} L^r_{p_i}$. Moreover, for all $x \in R$, $1 - x \leq e^{-x}$. So finally, we can write that $\delta_k \leq (1 - \frac{1}{n})^k \bigcup_{i=1}^{|P^*|} L^r_{p_i} \leq e^{-\frac{k}{n}} \bigcup_{i=1}^{|P^*|} L^r_{p_i}$. By substituting δ_k with $\bigcup_{i=1}^{|P^*|} L^r_{p_i} - \bigcup_{i=1}^{|P^k|} L^r_{p_i}$, rearranging and finally replacing $\bigcup_{i=1}^{|P^k|} L^r_{p_i}$ with its result L_k and $\bigcup_{i=1}^{|P^*|} L^r_{p_i}$ with L^*, we have: $L_k \geq (1 - e^{-\frac{k}{n}})L^*$ and for $l = k$ (lower bound) we have: $L_k \geq (1 - \frac{1}{e})L^*$. □

6 EXPERIMENTS

In this section, we test our approach on the sources presented in Section 4 and compare with the existing baselines.

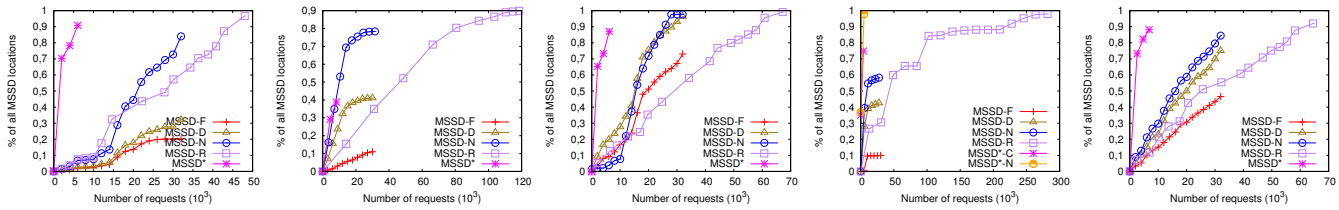
6.1 MSSD Experiments

We run *MSSD* algorithms using Krak as the seed source as it is the richest in terms of locations, points, and categories. We compare the results of the baseline (the initial locations of sources *SI*) to *MSSD-F* which uses a fixed radius of 2 km (Alg. 1), *MSSD-D* with a density-based approach to define the radius (Alg. 1a), *MSSD-N* with a nearest-neighbor method to define a flexible radius (Alg. 1b) and *MSSD-R* with a recursive method that starts with a radius of 16 km (the largest values accepted by all sources) and reduces the radius by a coefficient $\alpha = 2$ (Alg. 1c). Some APIs allow only an integer radius in the granularity of km, so $\alpha = 2$ is the smallest integer value accepted. Fig. 6 illustrates the improvement in the extracted data volume from each source by each version of *MSSD* over *SI*. Krak is not included since it is the *seed*. Google Places (GP) has the highest improvement of 98.4 times more locations extracted by *MSSD-R* compared to the initial ones from *SI*. Flickr had 4,084 locations initially, which become 4.3 times more with *MSSD-F* and above 9.5 times more with *MSSD-D*, *MSSD-N* and *MSSD-R*. In Foursquare (FSQ) and Yelp, *MSSD-F* extracts 3 and 2 times more locations respectively, but *MSSD-D*, *MSSD-N*, and *MSSD-R* retrieve up to 3.5. Twitter returns 10.7 times more with *MSSD-R* but still has a low number of locations overall. These values highlight that *in spite of their different scopes, all the sources relate to the same physical world*. *MSSD-R* performs the best with an improvement of 14.3 times more than *SI* but with extra requests that in the case of Twitter and GP can reach up to 8 times more than *MSSD-F*, *MSSD-D* and *MSSD-N*.

We ran the optimized version *MSSD** (Alg. 3) for each of the sources with initial radius of 16 km and initial $m = 10$ and $\epsilon = 500$ meters as parameters of DBSCAN. m, ϵ and r are recursively reduced by $\alpha = 2$. We compared *MSSD-F*, *MSSD-D*, *MSSD-N*, *MSSD-R* and *MSSD** regarding the number of requests performed and the locations retrieved. The results for each source are presented in Fig. 5. The number of requests is in the x-axis, whereas the number of locations is expressed as the percentage of the total of distinct locations extracted by all methods. *MSSD-R* provides the highest percentage of locations (above 95%) for all the sources but considerably more requests. For instance, for GP (Fig. 5b) and for Twitter (Fig. 5d), *MSSD-R* need respectively 3.8 and 8.7 times more requests than the *MSSD* versions with fixed number of requests (*MSSD-F*, *MSSD-D*, *MSSD-N*). For the same number of requests, *MSSD-N* provides a higher percentage of locations compared to *MSSD-F* and *MSSD-D* for all the sources. *MSSD** is the most efficient in terms of requests. For all sources except Google Places, *MSSD** gets around 90% of the locations with around 25% of the requests of *MSSD-F*, *MSSD-D* and *MSSD-N*. With regard to *MSSD-R*, *MSSD** uses 12%-15% of *MSSD-R* requests for Flickr, Yelp and Foursquare, 8.5% of *MSSD-R* requests for Google Places and 2.7% of *MSSD-R* requests for Twitter. In Google Places, *MSSD** can retrieve only 40% of the locations, because of its small result size of Google (Table 1). *MSSD-N* extracts 2 times more locations than *MSSD** but with 3 times more requests. In Twitter, *MSSD*-C* (with centroids) retrieves 20% more locations than *MSSD*-N* (with the nearest neighbor) using the same number of requests. To conclude, *MSSD** guarantees the best compromise for all the sources.

Setting α and radius r. In this experiment, we test different values of α and r for *MSSD**. When α is bigger, or r is smaller, fewer requests are performed, some areas are missed, and consequently, fewer locations are retrieved. Table 3 provides the trade-offs in terms of percentage of requests and percentage of locations of *MSSD** with regards to *MSSD-R* for each α (while fixing the radius at 16 km) and for each r (while fixing α at 2)(See Appendix B.2 in the full version of the paper [14] for details.). In all the cases, the additional requests of *MSSD** with small values of α 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=16km compared to r=1km. *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, 30 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



(a) Yelp (b) Google Places (c) Foursquare (d) Twitter (e) Flickr

Figure 5: Requests versus locations for different *MSSD* algorithms with Krak as seed

Sources	Req vs loc	Alpha								Radius				
		2	4	6	8	10	12	14	16	1	4	8	12	16
FSQ	% req	13.2%	13.0%	12.9%	12.9%	12.9%	12.9%	12.9%	12.9%	12.8%	12.9%	13.1%	13.3%	13.2%
	% loc	88.3%	78.1%	75.4%	74.8%	73.0%	71.5%	70.3%	69.5%	43.2%	82.9%	86.4%	88.1%	88.3%
Flickr	% req	15.8%	15.4%	15.2%	15.1%	15.0%	15.0%	15.0%	15.0%	13.2%	14.4%	15.3%	15.7%	15.8%
	% loc	96.5%	91.9%	86.9%	85.1%	82.9%	81.4%	80.3%	79.0%	49.1%	88.5%	94.8%	95.8%	96.5%
GP	% req	9.3%	9.0%	8.8%	8.7%	8.6%	8.5%	8.5%	8.4%	7.6%	9.0%	9.3%	9.3%	9.3%
	% loc	38.4%	34.3%	32.9%	32.4%	31.6%	30.9%	30.4%	30.1%	33.6%	37.2%	37.3%	38.2%	38.4%
Yelp	% req	17.5%	17.4%	17.4%	17.4%	17.4%	17.4%	17.4%	17.4%	17.3%	17.4%	17.4%	17.4%	17.5%
	% loc	98.2%	95.8%	93.7%	95.1%	93.4%	90.6%	89.8%	89.7%	51.1%	94.2%	97.7%	98.3%	98.2%
Twitter-C	% req	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%
	% loc	76.8%	58.6%	58.1%	57.1%	55.7%	52.8%	52.3%	52.3%	53.4%	61.5%	60.0%	58.9%	76.8%
Twitter-N	% req	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%
	% loc	99.5%	99.4%	99.0%	98.6%	98.3%	98.2%	97.9%	97.8%	86.4%	97.0%	97.1%	98.4%	99.5%

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

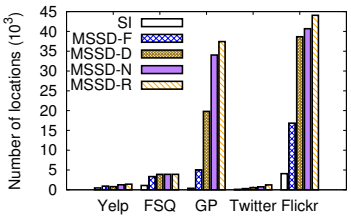
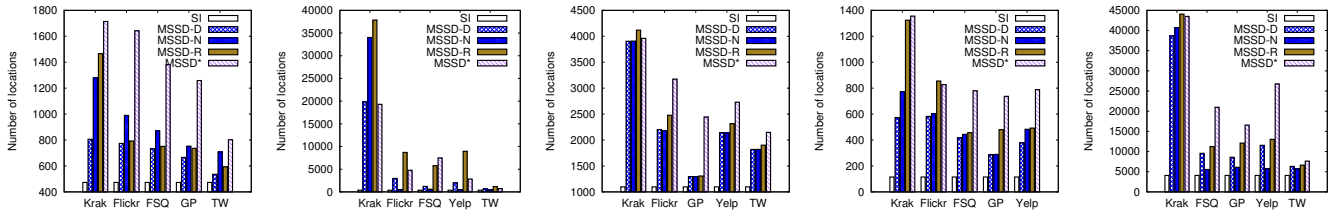
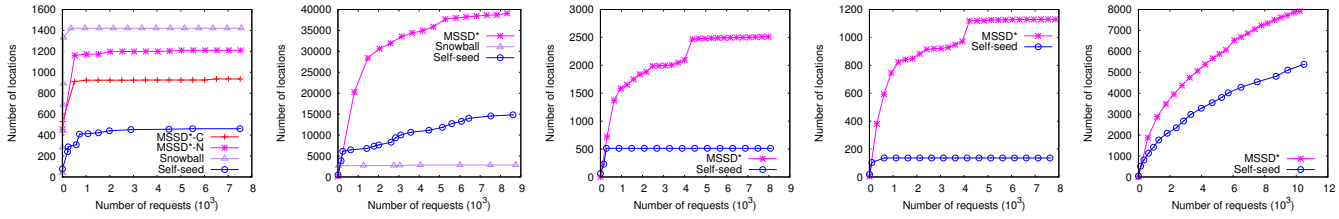


Figure 6: Nr of loc. per source



(a) Yelp (b) GooglePlaces (c) Foursquare (d) Twitter (e) Flickr

Figure 7: *MSSD* results with different seeds for all sources



(a) Twitter (b) Flickr (c) Foursquare (d) Yelp (e) Google Places

Figure 8: Number of request versus number of locations for *MSSD, *Snowball* and *Self-seed***

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

Source	Algorithm	Real 80%	Synthetic 20%	Req	Real 70%	Synthetic 30%	Req	Real 50%	Synthetic 50%	Req
Flickr	*MSSD-R*	85.62%	87.16%		84.44%	85.70%		82.71%	83.70%	
	*MSSD**	64.91%	66.22%	10.16%	63.59%	65.01%	10.12%	62.12%	64.28%	9.80%
Yelp	*MSSD-R*	97.17%	99.44%		97.10%	99.4%		97.10%	99.23%	
	*MSSD**	72.23%	77.81%	12.74%	68.89%	76.94%	12.49%	66.42%	72.59%	11.78%
FSQ	*MSSD-R*	94.27%	96.69%		94.19%	95.51%		94.07%	95.26%	
	*MSSD**	67,75%	74,57%	10.67%	66,15%	70,34%	10.62%	63,60%	68,65%	10.45%
GP	*MSSD-R*	92,42 %	78,76%		91,60%	76,15%		89,94%	69,33%	
	*MSSD**	34,48%	35,76%	13,85%	45,80%	33,59%	9,14%	38,09%	33,99%	13,49%
Twitter	*MSSD-R*	81,16 %	97,70%		81,16%	95,66%		80,92%	87,23%	
	MSSD-C*	45,30%	63,31%	2,45%	44,37%	62,09%	2,44%	48,88%	68,51%	2,45%
	MSSD-N*	69,79%	97,70%	2,53%	69,11%	94,51%	2,53%	60,90%	86,61%	2,53%

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

		Nr of loc.	Nr of users	Time period
Twitter	Snowball	1421	35	2015-2018
	Self-seed	461	101	2017-2018
	MSSD*-C	936	195	2017-2018
	MSSD*-N	1237	231	2017-2018
Flickr	Snowball	2885	46	2005-2018
	Self-seed	14910	1007	2005-2018
	MSSD*	39427	1740	2005-2018

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

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 Flickr 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 Flickr, 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**.

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 $\mathcal{D}$ of the locations by dividing the area in a grid of 1km x 1km and assigning each grid cell d a probability $p_d \sim \mathcal{D}_d$; 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

by Foursquare and Flickr with above 64%. What is more important, *MSSD-R and MSSD* are seen to be robust regardless of the ratio of synthetic to real data.* Although *MSSD** retrieves less than *MSSD-R*, this result is achieved using only around 10% of the requests of *MSSD-R*. In the case of Google Places, *MSSD** gets around 40% of the ground truth because of the small result size of only 20 locations per request. In the case of Twitter, *MSSD*-N* performs better than *MSSD*-C*.

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

[1] N. Armenatzoglou, S. Papadopoulos, and D. Papadias. 2013. A general framework for geo-social query processing. *PVLDB* (2013).
[2] N. Bennacer, F. Bugiotti, M. Hewasinghage, S. Isaj, and G. Quercini. 2017. Interpreting reputation through frequent named entities in twitter. In *WISE*.
[3] F. Burini, N. Cortesi, K. Gotti, and G. Psaila. 2018. The Urban Nexus Approach for Analyzing Mobility in the Smart City: Towards the Identification of City Users Networking. *Mobile Information Systems* (2018).
[4] J. L. Z. Cai, M. Yan, and Y. Li. 2016. Using crowdsourced data in location-based social networks to explore influence maximization. In *INFOCOM*.
[5] Z. Cheng, J. Caverlee, and K. Lee. 2010. You are where you tweet: a content-based approach to geo-locating twitter users. In *CIKM*.
[6] E. Cho, S. A. Myers, and J. Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *KDD*.
[7] Tobias Emrich, Maximilian Franzke, Nikos Mamoulis, Matthias Renz, and Andreas Züfle. 2014. Geo-social skyline queries. In *DASFAA*.
[8] M. Ester, H. Kriegel, J. Sander, X. Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *KDD*.
[9] S. Feng, X. Li, G. Zeng, Y.and Cong, Y. Meng Chee, and Q. Yuan. 2015. Personalized Ranking Metric Embedding for Next New POI Recommendation.. In *IJCAI*.
[10] G. Ference, M. Ye, and W. Lee. 2013. Location recommendation for out-of-town users in location-based social networks. In *CIKM*.
[11] H. Gao, J. Tang, X. Hu, and H. Liu. 2013. Exploring temporal effects for location recommendation on location-based social networks. In *RecSys*.
[12] H. Gao, J. Tang, X. Hu, and H. Liu. 2015. Content-Aware Point of Interest Recommendation on Location-Based Social Networks.. In *AAAI*.
[13] D. Hristova, M. J. Williams, M. Musolesi, P. Panzarasa, and C. Mascolo. 2016. Measuring urban social diversity using interconnected geo-social networks. In *WWW*.
[14] S. Isaj and T. B. Pedersen. 2019. Seed-Driven Geo-Social Data Extraction - Full Version. *CoRR* (2019).
[15] D. Jurgens. 2013. That's What Friends Are For: Inferring Location in Online Social Media Platforms Based on Social Relationships. *ICWSM* (2013).
[16] D. Jurgens, T. Finethy, J. McCorriston, Y. T. Xu, and D. Ruths. 2015. Geolocation Prediction in Twitter Using Social Networks: A Critical Analysis and Review of Current Practice. *ICWSM* (2015).
[17] H. Kwak, C. Lee, H. Park, and S. Moon. 2010. What is Twitter, a social network or a news media?. In *WWW*.
[18] Ankita L., Srikanta B., and Deepak P. 2017. LoCaTe: Influence Quantification for Location Promotion in Location-based Social Networks. In *IJCAI-17*.
[19] R. Lee and K. Sumiya. 2010. Measuring geographical regularities of crowd behaviors for Twitter-based geo-social event detection. In *GIS-LBSN*.
[20] S. H. Lee, P. Kim, and H. Jeong. 2006. Statistical properties of sampled networks. *Physical Review E* (2006).
[21] G. Li, S. Chen, J. Feng, K. Tan, and W. Li. 2014. Efficient location-aware influence maximization. In *SIGMOD*.
[22] J. Li, T. Sellis, J. S. Culpepper, Z. He, C. Liu, and J. Wang. 2017. Geo-social influence spanning maximization. *TKDE* (2017).
[23] L. Li, M. F. Goodchild, and B. Xu. 2013. Spatial, temporal, and socioeconomic patterns in the use of Twitter and Flickr. *CaGIS* (2013).
[24] N. Li and G. Chen. 2009. Analysis of a location-based social network. In *CSE*.
[25] Y. Liu, T. N. Pham, G. Cong, and Q. Yuan. 2017. An experimental evaluation of point-of-interest recommendation in location-based social networks. *PVLDB* (2017).
[26] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. 1978. An analysis of approximations for maximizing submodular set functionsâĂŤI. *Mathematical programming* (1978).
[27] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo. 2012. A random walk around the city: New venue recommendation in location-based social networks. In *SocialCom/PASSAT* .
[28] D. Preoţiuc-Pietro and T. Cohn. 2013. Mining user behaviours: a study of check-in patterns in location based social networks. In *WebSci*.
[29] M. A. Saleem, R. Kumar, T. Calders, and T. B. Pedersen. 2018. Effective and efficient location influence mining in location-based social networks. *Knowledge and Information Systems* (2018).
[30] M. A. Saleem, R. Kumar, T. Calders, X. Xie, and T. B. Pedersen. 2017. Location Influence in Location-based Social Networks. In *WSDN*.
[31] S. Scellato, C. Mascolo, M. Musolesi, and V. Latora. 2010. Distance Matters: Geo-social Metrics for Online Social Networks.. In *WOSN*.
[32] S. Scellato, A. Noulas, and C. Mascolo. 2011. Exploiting place features in link prediction on location-based social networks. In *KDD*.
[33] H. Wang, M. Terrovitis, and N. Mamoulis. 2013. Location recommendation in location-based social networks using user check-in data. In *SIGSPATIAL/GIS*.
[34] M. Weiler, K. A. Schmid, N. Mamoulis, and M. Renz. 2015. Geo-social co-location mining. In *GeoRich@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*.
[36] F. Yu, N. Che, Z. Li, K. Li, and S. Jiang. 2017. Friend recommendation considering preference coverage in location-based social networks. In *PAKDD*.
[37] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. 2012. Evaluating geo-social influence in location-based social networks. In *CIKM*.
[38] J. Zhang and C. Chow. 2013. iGSLR: personalized geo-social location recommendation: a kernel density estimation approach. In *SIGSPATIAL/GIS*.
[39] S. Zhao, T. Zhao, H. Yang, M. R. Lyu, and I. King. 2016. STELLAR: Spatial-Temporal Latent Ranking for Successive Point-of-Interest Recommendation.. In *AAAI*.
[40] Q. Zhu, H. Hu, C. Xu, J. Xu, and W. Lee. 2017. Geo-social group queries with minimum acquaintance constraints. *VLDB J.* (2017).