2014 IEEE 15th International Conference on Mobile Data Management



# A Clustering Approach to the Discovery of Points of Interest from Geo-Tagged Microblog Posts

Anders Skovsgaard[#] Darius Šidlauskas[#] Christian S. Jensen[*]

[#]*Aarhus University* {anderssk, dariuss}@cs.au.dk
[*]*Aalborg University* csj@cs.aau.dk

***Abstract*—Points of interest (PoI) data serves an important role as a foundation for a wide variety of location-based services. Such data is typically obtained from an authoritative source or from users through crowdsourcing. It can be costly to maintain an up-to-date authoritative source, and data obtained from users can vary greatly in coverage and quality. We are also witnessing a proliferation of both GPS-enabled mobile devices and geo-tagged content generated by users of such devices. This state of affairs motivates the paper's proposal of techniques for the automatic discovery of PoI data from geo-tagged microblog posts. Specifically, the paper proposes a new clustering technique that takes into account both the spatial and textual attributes of microblog posts to obtain clusters that represent PoIs. The technique expands clusters based on a proposed quality function that enables clusters of arbitrary shape and density. An empirical study with a large database of real geo-tagged microblog posts offers insight into the properties of the proposed techniques and suggests that they are effective at discovering real-world points of interest.**

## I. Introduction

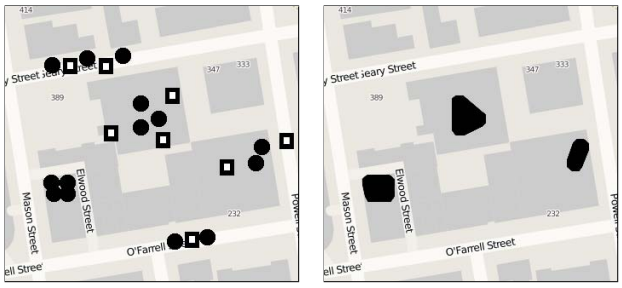
Points of Interest (PoIs) represent geographical entities (e.g., restaurants, museums, hotels), as a series of geo-referenced points with attached semantics (e.g., names or addresses). PoIs are important for a wide variety of services, e.g., Google/Yahoo/Bing Maps, Yelp, TripAdvisor. Also, they have high value for companies like Factual and Google, who sell PoIs. PoIs can represent formal locations defined by authoritative sources of information (e.g., lists of landmarks and buildings for tourist attractions), or they can represent informal gatherings that result from local activities (e.g., places where demonstrations occur, picnic spots, public events). PoIs can be static (e.g., the Empire State Building) or changing (e.g., a specific exhibition at a museum). Also, PoIs can be relevant only during some time (e.g., during a festival).

PoIs may be collected in different ways that each has its shortcomings. Manual PoI collection and maintenance is expensive and affordable only to big corporations. It may be possible for PoI owners to contribute information themselves; however, it is likely that they only contribute to the most popular services and only if it is beneficial to them. Informal gatherings may be more difficult to capture due to the lack of knowledge about these or the absence of incentives. Collaborative projects such as OSM[1] depend on volunteered, crowdsourced data, and thus coverage and quality vary greatly.

[1]http://www.openstreetmap.org

We leverage the increasing availability of geo-tagged microblog posts to enable a new way of obtaining PoIs. Increasingly large volumes of geo-tagged content are becoming available with the proliferation of GPS-enabled mobile devices. Specifically, users of such devices are creating large amounts of geo-tagged microblog posts describing their daily activities. Some users may report on information from their current location. For example, the circles in Figure 1a represent geo-tagged posts containing the term "hotel." Other users in the same region may report on information unrelated to the location (the square symbols in the figure). We aim to extract PoIs from such user-generated content.

Intuitively, users are likely to use specific textual descriptions more often in some regions than in others. For example, posts mentioning "hotel" might be more frequent in a region containing a hotel than in a region containing a supermarket. We assume a PoI can be created for a location if a sufficiently high percentage of the posts in the region of the location have a similar textual description. An example result is shown in Figure 1b, where three clusters are formed from the objects in Figure 1a. When sufficiently many dissimilar posts exist in the same region, no cluster may be formed. Due to the nature of geo-tagged posts, the clusters may be of arbitrary shape and density.




(a) Geo-Tagged Microblog Posts (b) Discovered "Hotel" Clusters

Fig. 1: Clustering of Real Objects

A system that enables the creation of PoIs from geo-tagged microblog posts should satisfy several challenging requirements.

First, due to the diverse nature of spatial objects, the clusters may be of arbitrary shape. Second, in addition to the shapes

978-1-4799-5705-7/14 $31.00 © 2014 IEEE
DOI 10.1109/MDM.2014.28

being arbitrary, clusters can overlap, and a cluster can span several other clusters. Third, there can be significant amounts of irrelevant data in a region. A clustering algorithm therefore must be robust to noise or outliers (we do not distinguish between these two). Fourth, different clusters may have very different densities, and the density of posts inside a cluster representing a PoI may also vary. Fifth, a spatial clustering algorithm is required that takes into account not only the spatial proximity among posts, but also the similarity among their textual attributes. Sixth, since geo-tagged content is obtained in arbitrary order, the results of a good clustering approach should be independent of the ordering of the input data. In other words, it should be order-insensitive with respect to the input data.

To meet these requirements, we contribute a new type of clustering method for spatio-textual objects, termed CLUSTO (clustering of spatio-textual objects). Specifically, this method satisfies the following requirements:

1) Discovers clusters of arbitrary shape.
2) Enables overlapping clusters.
3) Is robust to noise and outliers.
4) Identifies clusters of varying density.
5) Takes into account both spatial and textual attributes.
6) Is insensitive to the ordering of the input data.

Many previous works have explored research directions related to microblogs [1]–[4], [6], [11]–[13], [16], [18]–[22]. However, to the best of our knowledge, there is no existing work that address the discovery of PoIs using geo-tagged microblog posts. Spatial data mining is a popular research area (see Section V), where the most related work includes density-based clustering algorithms [5], [7]–[9], [17]. However, they fail to fulfill requirement 5, which is essential. In the experimental evaluation, we extend DBSCAN to support non-spatial attributes, but show that it suffers from over-expansion.

CLUSTO is based on nearest neighbor chaining that fulfills requirements 1 and 6. The proposed quality-based clustering criterion solves, in combination with the nearest neighbor chaining, requirements 3, 4, and 5. CLUSTO allows overlapping clusters, depending on the setting of a quality-threshold (addressing requirement 2).

In the paper's empirical study, we study the effectiveness of CLUSTO using a real-world dataset. By comparing with Google Places, we show that CLUSTO is able to successfully identify most PoIs from an abundance of geo-tagged tweets (we have accumulated more than 280,000 geo-tagged tweets from downtown San Francisco from the Twitter stream during a period of more than a year).

The remainder of the paper is structured as follows. Section II proposes the notion of quality based clustering. The proposed solution along with two base-lines are presented in Section III. The experimental evaluation is given in Section IV. Section V covers related work. Finally, we conclude in Section VI.

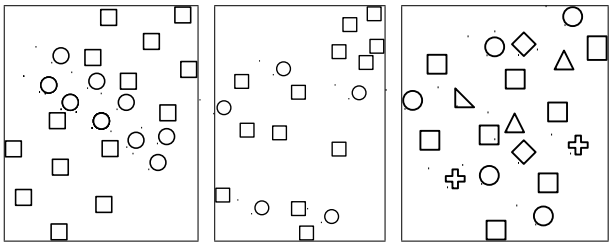
## II. Quality Based Clustering

A spatio-textual object has a spatial position and a textual description. An object may have a unique textual description, or parts of its description may be shared with other objects. Many different textual descriptions may exist in a dataset of spatio-textual objects, and we represent objects with different descriptions by different shapes in our examples. In Figure 2, squares may represent objects with text "hotel," while circles may represent objects with text "park."

Spatially nearby objects do not necessarily have similar textual descriptions. For example, consider the objects in Figure 2a. Although their spatial distribution or density is very similar, one can easily identify three clusters: two clusters of squares and one cluster of circles.

Objects that share a textual description may have neighboring objects with different textual descriptions. An example is provided in Figure 2b, where circles occur arbitrarily among the squares. The objects in this example may be regarded as a single cluster of squares with a small amount of noise. Also, the spatial distance between objects may vary greatly within a single cluster. For example, there may be a higher concentration of objects in a specific part of a park, but more sparse regions of the park may still be considered as part of the park.

When there is much variation in the textual description of the objects, it may not be possible to determine the type of the cluster. In this case, all the objects will be considered as noise. For example, in Figure 2c, there is no clear dominating textual description among the objects.



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

Fig. 2: Sample Datasets where Shapes Represent Different Textual Descriptions

To simplify the problem of clustering spatio-textual objects, each textual description may be considered in an isolated manner. That is, clustering is performed for each set of objects that share a textual description in isolation from objects with different textual descriptions. As a result, clustering is based only on the spatial dimension. Many spatial clustering algorithms exist that take into account only the spatial dimension (see Section V). However, this approach implies that the textual information is ignored and that clusters may expand over unrelated but spatially nearby objects. For example, when only the squares are considered in Figure 2a, the natural cluster is all the squares. As the spatial distribution among the square

objects is similar, it becomes difficult to distinguish the actual clusters. When considering the circles, they clearly separate the two clusters of squares. Also, when textual descriptions are ignored, a set of noisy objects may form a false cluster. For example, the objects in Figure 2c may result in multiple clusters because the textual descriptions are considered in isolation. Therefore, purely spatial clustering is not adequate in our targeted setting.

Motivated by this, we aim for an approach that takes into account both textual and spatial attributes of an object and is able to detect the above clusters correctly. We continue to formalize the problem.

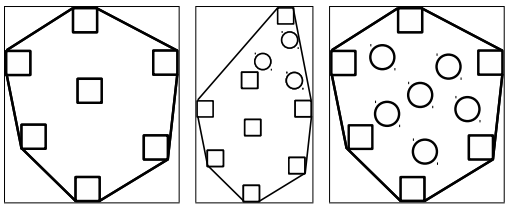
Clustering spatio-textual objects results in a set of objects which has a spatial region. The region may be defined using different techniques. In this paper, for ease of understanding, we define the region of a cluster to be the convex hull of all objects in the cluster. An example is provided in Figure 3a. Other techniques for enclosing the objects, like the orthogonal convex hull, the minimum bounding box, or the bounding sphere, may be used with the proposed solution.

*Definition 1.* Let $D$ be a set of objects. Given a set of objects $O$ in $D$, function $\gamma$: $D \rightarrow D$ returns the minimal subset of $O$ that has the same convex hull as $O$.

The objects in the enclosing region, regardless of their textual description, are called `ClusterEnclosed`.

*Definition 2.* Let $D$ be a database of objects. Given a set of objects $O$ in $D$, function $\Lambda$: $D \rightarrow D$ returns the set of objects in $D$ that are enclosed by the convex hull of $O$. We say that the objects returned by $\Lambda(O)$ are `ClusterEnclosed` by $\gamma(O)$.

An example of a set of `ClusterEnclosed` objects is provided in Figure 3b. The set $O$ contains the squares, $\gamma(O)$ contains the 6 squares on the border, and $\Lambda(O)$ contains all 11 objects inside the convex hull. Note that the three circles do not contain the same textual description as the squares, but are still `ClusterEnclosed` objects.



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

Fig. 3: Sample Datasets where Shapes Illustrate Different Textual Descriptions

The addition of a single object to a given cluster may increase the number of `ClusterEnclosed` objects by more than one. The new objects that become `ClusterEnclosed` may or may not share the same textual description as the other objects in the set. In Figure 3b, three objects (circles) were added to `ClusterEnclosed` when the top square object was included.

We proceed to define the quality of a given textual description, termed *annotation*, in a set of `ClusterEnclosed` objects. Ideally, a set of `ClusterEnclosed` objects is preferred, where all objects share the same textual description, as in Figure 3a.

Any cluster can easily be annotated with a textual description if the `ClusterEnclosed` objects all share the same textual description. However, this may not always be the case. As seen in Figure 3b, objects with different textual description may occur. These "noise" objects may be acceptable since one cannot expect that all objects in a given region contain the same textual description. For example, in a park, some may publish the information that they are actually in the park, while others may publish information about arbitrary topics. To accommodate this, we introduce means of measuring the quality of an annotation of a cluster.

Intuitively, a cluster like the one in Figure 3c is of lower quality than the cluster in Figure 3a. Figure 3a has seven objects that all share the same textual description, making the annotation of the cluster obvious. However, Figure 3c has six objects for each of two textual descriptions, making it difficult to determine a good annotation of the cluster. We want to reflect this in our quality function. The quality function depends on the number of noise objects and the number of objects with a textual description that contains the annotation, $a$. Thus, a smaller fraction of noise objects results in a better quality of the cluster. More formally:

*Definition 3.* The quality of an annotation $a$ of a given set of objects $S$ is given by:

$$q(S, a) = \frac{|S_a|}{|S|}, \tag{1}$$

where $S_a$ is the objects in the set with a textual description containing $a$. The function returns a value in [0, 1], and the value 1 represents the best possible quality.

**Example 1.** Consider the clusters in Figures 3b and 3c. The quality of each cluster with respect to the square annotation is $8/11$ and $6/12$, respectively. Clearly, the cluster in Figure 3b has a better quality since it is dominated by square objects.□

When a cluster is expanded by a new object or set of objects, the quality of the cluster must be recomputed. Adding an object to a cluster with a large number of objects has minimal impact on the quality score. Consider a cluster like the one in Figure 3a with a large number of objects and a quality of 1. Adding an object to this cluster may also produce a good quality score, even with a number of accompanying noise objects as seen from the example in Figure 3b. Therefore, large clusters may expand excessively and may include numerous noise objects. Large clusters with initially high quality are in particular likely to suffer from this over-expansion.

To avoid expansions that decreases the quality of the resulting clusters, we introduce a new measure of cluster quality when expanding clusters. The expansion of a cluster involves merging two clusters that each consists of one or more objects. The number of noise objects resulting from the merge may

have different impact on the quality when considering the clusters individually. Therefore, we define the *merged* quality of two clusters to be determined by the lower quality computed for each cluster in combination only with newly added (due to merge) objects.

*Definition 4.* The quality of a cluster with respect to an annotation $a$ when merging two clusters, $c_1$ and $c_2$ is given by:

$$mergedQ(c_1, c_2, a) = \min_{x \in \{c_1, c_2\}} q(\Lambda(c_1 \cup c_2) \setminus \Lambda(x), a) \quad (2)$$

**Example 2.** We consider the clusters in Figure 4 and calculate the merged quality with respect to the square annotation. The 3 top square objects are cluster $c_1$ and the lower 7 objects are cluster $c_2$. When combining the two clusters, the merged quality is:

$$mergedQ(c_1, c_2, \square) = \min\{\frac{3}{6}, \frac{7}{10}\} = 0.5 \quad (3)$$

Note that the quality of the complete cluster exceeds the merged quality. In this example the quality of the complete cluster is: $\frac{10}{13}$. □

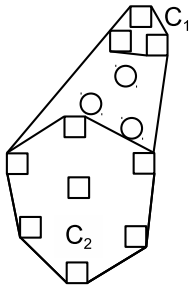



Fig. 4: Two Clusters Separated by Three Noisy Objects

### A. Candidate Annotations

A cluster consists of a number of objects that share a textual description, an annotation, and a number of noise objects. To find annotations that can form a cluster, we consider multiple combinations of the terms in the textual descriptions. We consider each single-term text and a multi-term text starting with capital letters as a candidate annotation. Thereby, we aim to capture places of interest and their descriptive terms.

**Example 3.** Table I lists all possible annotations for the textual description: "We are having a barbecue in Central Park." Note that stop words are removed. □

In the following, we consider clustering using only the above candidate annotations. Notably, many other approaches can be supported, e.g., all term combinations or all $n$-grams can be derived from textual descriptions. We also ignore the problem of lexical heterogeneity, i.e., when different textual descriptions refer to the same real-world objects (e.g., Museum of Modern Art versus MoMA). While we expect this to improve the quality of the clustering, such studies are beyond the scope of this paper.

| *annotation* |
|---|
| ⟨ barbecue ⟩ |
| ⟨ Central ⟩ |
| ⟨ Park ⟩ |
| ⟨ Central Park ⟩ |

TABLE I: Example of Annotations

## III. Proposed Solution

We proceed to present the CLUSTO clustering technique. CLUSTO uses the above-defined merged cluster quality concept and is configured with two parameters. The first parameter, $minQ$, defines the threshold for the $mergedQ$ value that controls when two clusters are merged. Two clusters $c_1$ and $c_2$ with annotation $a$ are merged if and only if $mergedQ(c_1, c_2, a) \geq minQ$. The second parameter, $minSize$, defines the minimum number of objects that a valid cluster must contain.

We start by giving a general description of how our quality-based function can be integrated into cluster expansion algorithms, including the approach taken by CLUSTO. Then we provide algorithm implementation details. We assume that each object is also a single-object cluster, and thus we use simply the term "cluster" in the following.

### A. Spatial Cluster Expansion

There are several approaches to how $mergedQ$ can be integrated into spatial cluster expansion.

**Range-based Expansion.** Motivated by DBSCAN [5] and its cluster expansion based on a configured maximum range parameter ($\epsilon$), a similar approach can be taken to find clusters using the quality function. A random cluster is selected, and a range search with distance $\epsilon$ around it is performed. The cluster is merged with each of the clusters in the range if the merged quality of the resulting clusters ($mergedQ$) exceeds the given threshold ($minQ$).

In addition to it being difficult to choose $\epsilon$ optimally, this approach suffers from the same problem as DBSCAN—varying density. Consider the example dataset in Figure 2a, where the distances between the square objects are similar. In this case, it may be possible to select a suitable value for $\epsilon$. However, the distances between the objects in Figure 2b vary greatly. Therefore, to capture all clusters, the value must be set high in order to contend with objects that are far apart. This may result in large clusters since the quality is evaluated for all objects in the range. Thus, this approach fails to detect smaller clusters.

**Nearest Neighbor Expansion.** This approach expands a cluster based on its nearest neighbor (NN) cluster. Initially, a random cluster is selected, and an NN search is performed to find its NN cluster. As before, the two clusters are merged if the merged quality exceeds the given threshold. As such,

this approach overcomes the varying-density problem of the previous approach. However, it is now sensitive to the order in which the clusters are considered for merging. Since the initial cluster is selected at random for each candidate annotation, the final clustering may vary from run to run.

In CLUSTO, we overcome the shortcomings of both approaches as follows.

**Reciprocal Nearest Neighbor Expansion.** Our approach builds on nearest neighbor (NN) chains [14]. An NN-chain consists of an arbitrary cluster, followed by its NN, which in turn is followed by its NN from the remaining clusters, and so on. Such a chain ends in a mutual or reciprocal (RNN) pair, i.e., a pair of clusters $c_1$ and $c_2$ such that the NN of $c_1$ is $c_2$, and vice versa. When such an RNN pair is found, the corresponding clusters are merged. Since such an expansion between two clusters is performed only if they have no other closer neighbors, the algorithm can recursively perform an agglomerative hierarchical clustering until all clusters are merged into a single (root) cluster. We employ single-linkage clustering. As discussed above, we extend this approach with a stopping condition based on the merged-quality function.

Since the next candidate is the nearest neighbor, it does not require any range parameters ($\epsilon$) and is immune to varying density.

*1) Correctness:* Using the NN-chaining algorithm, the resulting merged clusters are not affected by the order in which the clusters are selected for merging. That is, the algorithm eventually arrives at the same clustering independently of the order in which the clusters are traversed. The correctness relies on the reducibility property [14].

**Lemma.** The NN-chaining algorithm forms the same clusters independent of the ordering of the dataset.

**Proof:** We verify the reducibility property of the NN-chaining algorithm, implying that when two clusters are merged, they do not affect any other clusters.

Given two RNN clusters $i$ and $j$, and any other cluster $c$, using single-linkage clustering, there is some (Euclidean distance) $p$ satisfying the following:

$$||i, j|| < p,\ ||i, c|| > p, \text{ and } ||j, c|| > p.$$

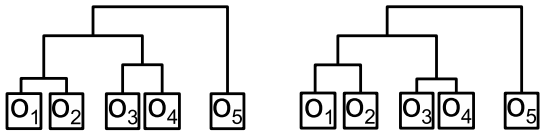
With single-linkage clustering, no object is closer to $i$ than $j$ and vice versa. Thus, the following holds when merging $i$ and $j$:

$$||i + j, c|| > p.$$

This verifies the reducibility property. Thereby, $i$ and $j$ can be merged without effecting the RNN properties of any other clusters. □

As a result of using a clustering algorithm that satisfies the reducibility property, the order in which clusters are discovered may vary, but the eventual clustering is the same. For example, consider the two dendrograms in Figure 5. The dendrogram in Figure 5a may have either object $o_1$ or $o_2$ as a starting point, whereas the dendrogram in Figure 5b may have any of $o_3$, $o_4$, or $o_5$ as a starting point. As seen from the figures, the order in which the clusters are formed may change depending on the starting point: however, the formed clusters are the same.

Intuitively, the order enforced by RNN expansion is favored because it first tries to merge the clusters that are closest to each other. This is confirmed by our empirical study (in Section IV).




(a) Dendrogram 1 (b) Dendrogram 2

Fig. 5: Sample Dendrograms where the Clustering is not Impacted by the Starting Point

This approach can handle excessive expansion of clusters and noise in the dataset. For example, consider the objects in Figure 6. Two clusters may be formed: one cluster with the five left-most objects and one cluster with the objects $o_1, o_2, o_3$, and $o_4$. With the reciprocal nearest neighbor requirement, object $o_5$ is regarded as noise since its NN is the cluster of four (and merging them would result in a much lower quality of the cluster). Similarly, such a stopping condition may prevent the inclusion of the upper-right object in Figure 3b.

When a cluster fails to expand to its NN, it is considered complete. Since the nearest neighbor sharing the same partial textual description is the most related, no other object may be more relevant for expansion. Continuing the example in Figure 6, the cluster of four will not include any of the five left-most objects. Intuitively, if the cluster should be expanded, there would be objects in that direction with shorter distance than the distance to the outlier $o_5$. Therefore, when we encounter a too low quality the first time, we stop and avoid over-expansion of the cluster.


Fig. 6: Sample Dataset where Cluster Expansion is Stopped

### *B. CLUSTO Implementation*

We proceed to describe the implementation of CLUSTO. The following algorithms integrate the selection of candidate annotations, nearest-neighbor chaining, and the stopping rule based on the proposed merged-quality function. The algorithms are designed to run on a large dataset of spatio-textual objects (such as geo-tagged microblog posts) for effective PoI identification.

*1) Clustering Algorithm:* Algorithms 1–3 show the clustering approach implemented in CLUSTO. The entry point is Algorithm 1 with the global parameters configured at the beginning (Lines 1–3). The for loop iterates through all candidate annotations. The annotations are sorted descendingly

by the length of their textual description so that the most descriptive/specific (i.e., longest, probably containing multiple terms) annotations are considered first. Each annotation is passed to Algorithm 2 that performs cluster expansion.

**Algorithm 1:** `CLUSTO`(Dataset $D$)

```
   /* Global variables: */
1  minSize ← minimum cluster size;        // config. parameter
2  minQ ← quality threshold;              // config. parameter
3  V ← dictionary to store clustering;    // {string→clusters}
4  sort D by text length in descending order;
5  foreach Annotation a ∈ D do
6      V[a] ← ExpandCluster(a, D);
7  return V;
```

Algorithm 2 is responsible for expanding all possible clusters for the given annotation. It starts by initializing five working variables (Lines 1–5): one stack ($A$), two lists of clusters ($R$ and $E$), and two sets of objects ($D_a$ and $V_a$). The stack maintains clusters that may be further expanded when an RNN is found. The two lists of clusters contain the clusters that are no longer considered for expansion. Thus, the clusters in the list $R$ are completed and may be returned at any time. The set $D_a$ stores all candidate objects for clustering with the given annotation, i.e., objects with textual description containing $a$. The set $V_a$ stores objects that already formed valid clusters containing the given annotation.

After the initialization, a while loop follows that iterates through all unvisited objects containing the given annotation. Each object is randomly selected, marked as visited, and pushed to the stack of active clusters (Lines 7–9). Then, the processing continues in the inner while loop until the stack of active clusters is empty (Lines 10–25).

The loop follows the logic behind NN-chaining, taking into account the stopping condition. A cluster ($c$) is popped out from the stack, and NN search is performed (Line 13). We use the Euclidean distance calculated from an object in $c$ to its NN object. If the found NN object does not belong to any cluster, $c_{nn}$ stores a single-object cluster. Otherwise, $c_{nn}$ stores the cluster that the found object belongs to. Note that NN search excludes the objects from already formed clusters containing the same annotation ($V_a$). Since longer annotations are considered first, later clustering with a shorter sub-annotation does not reuse the same objects. For example, objects that formed "Central Park" are not reused when clustering "Park."

Next, the algorithm performs NN-chaining, taking into account the stopping condition based on the quality function. The NN of $c$ is examined as follows. If no NN to $c$ can be found or the found NN belongs to the list of previously rejected or accepted clusters (Line 14), the expansion of $c$ is stopped by calling Algorithm 3 (Line 15), which finalizes the clustering of $c$ (more on that in a moment). If the found NN and $c$ are reciprocal (Line 17), the merged quality of the clusters is computed and compared to threshold $minQ$ (Line 18). If the quality is satisfactory, $c$ is expanded with its NN cluster

**Algorithm 2:** `ExpandCluster`(Annota. $a$, Dataset $D$)

```
1  A ← stack of active clusters;
2  R ← list of approved clusters;
3  E ← list of rejected clusters;
4  D_a ← {o ∈ D | o.text contains a};
5  V_a ← {o ∈ c | c.a contains a, c ∈ V};
6  while D_a contains non-visited objects do
7      select random unvisisted object, o ∈ D_a;
8      mark o as visited in D_a;
9      push {o} to A;                  // as a single-object cluster
10     while A is non-empty do
11         c ← pop cluster from A;
12         c_nn ←NN cluster of c in D_a \ (c ∪ V_a);
13         mark c_nn as visited;
14         if c_nn is null or c_nn ∈ (E ∪ R) then
               /* Stop expanding - no more valid NNs */
15             FinishCluster(c, A, R, E);
16             continue;
17         if c_nn ∈ A then                // RNN pair found
18             if mergedQ(c, c_nn, a) ≥ minQ then
19                 remove c and c_nn from A;
20                 push (c ∪ c_nn) to A;   // merge and push
21             else    // Stop expanding - the quality is too low
22                 FinishCluster(c, A, R, E);
23                 FinishCluster(c_nn, A, R, E);
24         else                            // not RNN
25             push c_nn to A;
26 return R;
```

by removing them from the stack of active clusters (Line 19) and pushing the newly merged cluster instead (Line 20). As such, the pushed cluster may be further expanded in the next iteration. Otherwise, the expansion of both $c$ and its NN are finalized by calling Algorithm 3 as before (in Lines 22 and 23, respectively).

Finally, the found NN may not be an RNN to $c$. In this case, it is simply pushed on the stack of active clusters (Line 25), implying that the search of its RNN is performed in the next iteration.

Algorithm 3 performs the final steps for the given cluster. First, if the cluster is large enough, it is added to the list of approved clusters ($R$). Otherwise, it is added to the list of rejected clusters ($E$). In either case, all the objects in the cluster are marked as visited. This implies that the objects are no longer considered for clustering, as they formed or failed to form a cluster under the given annotation. Lastly, the cluster is removed from the stack of active clusters ($A$).

*2) Complexity Analysis:* Each object from $D_a$ is added only once to the stack of active clusters (Line 9), since it is marked as visited on entry, and, when it is removed it is either added to the list of approved or rejected clusters. Therefore, it may never enter the stack again as seen from Lines 7 and 14. This

**Algorithm 3:** `FinishCluster`(Cluster $c$, Stack $A$, List $R$, List $E$)

```
1 if |c| ≥ minSize then
2     R ← R ∪ c;
3 else
4     E ← E ∪ c;
5 remove c from A;
```

gives the space consumption in the stack of $O(|D_a|)$, for each annotation in $D$. The NN-chain is grown from the most recent cluster in the stack, and a cluster may merge with as many as $|D_a|-1$ clusters. This gives a run-time of $O(|D_a|^2)$, for each annotation in $D$ [14].

In the following example, we walk through the entire clustering process in CLUSTO.

**Example 4.** We run the algorithm with the 8 objects seen to the right in Figure 6 as the dataset and set the minimum quality threshold, $minQ$, to 0.5 and the minimum cluster size, $minSize$, to 2. We assume that the $\square$ annotation is longer and thus select a random starting point $o_2$ (in Algorithm 2, Line 7). As such, the first single-object cluster is pushed to $A$.

$A = \{\{o_2\}\}$

In Line 13, we find that the NN to $o_2$ is $o_1$. Since $o_1$ does not exists in $A$, it is pushed to $A$ too (Line 25).

$A = \{\{o_2\}, \{o_1\}\}$

Next, the NN to object $o_1$ is object $o_2$. Since $o_2$ exists in $A$, they are RNNs, and the merged quality is computed (Line 18). The quality of the two objects is 1, since no other `ClusterEnclosed` objects exist. The quality exceeds the threshold, and the objects are merged and added to $A$.

$A = \{\{o_1, o_2\}\}$

The expansion continues:

$A = \{\{o_1, o_2\}, \{o_3\}\}$

$A = \{\{o_1, o_2\}, \{o_3\}, \{o_4\}\}$

$A = \{\{o_1, o_2\}, \{o_3, o_4\}\}$

$A = \{\{o_1, o_2, o_3, o_4\}\}$

$A = \{\{o_1, o_2, o_3, o_4\}, \{o_5\}\}$

The process stops when $o_5$ becomes the RNN of the cluster of the other four objects. The merged quality is $mergedQ(\{o_1, o_2, o_3, o_4\}, \{o_5\}, \square) = 0.25$, which is below the threshold $minQ = 0.5$. Thus, the clusters are not expanded any further as seen from the two calls to Algorithm 3. The cluster of four is added to the set of approved clusters, $R$, while the cluster consisting of $o_5$ is added to the list of rejected clusters, $E$.

The order in this example follows the order of the dendrogram in Figure 5a. Had $o_3$, $o_4$, or $o_5$ been the random starting point, the order would have followed the dendrogram in Figure 5b, but it would have produced the same clustering. $\square$

## IV. Experimental Evaluation

We empirically evaluate CLUSTO's effectiveness at discovering points of interests using a real dataset and compare it against the DBSCAN algorithm. First, we present our experimental setup. Then, we describe the results of a number of experiments.

### A. Experimental Setup and Data

We use a commodity machine with a quad-core Intel Core i5-2520M (2.5 GHz) processor and 8 GB of main memory. In all experiments, the data is loaded into PostgreSQL with geo-spatial indexes capable of measuring distances on the Earth's surface. All data in the database is stored on disk. The proposed solution is implemented in a single-threaded Java application. All possible annotations are pre-processed and stored in main memory using an inverted file index (a mapping from textual annotation to the corresponding object identifiers in the database).

To evaluate the quality of the computed clusters, we utilize the Google Places API[2] that enables us to retrieve place information (including points of interests) within a given geographic region.

We collected all geo-tagged messages from the public Twitter FireHose in the period from September, 2012 to November, 2013. For this study, we extracted all tweets issued within a 6 km$^2$ down-town region in San Francisco, USA. In our micro-benchmarks, we found the region to be very well covered by Google Places. The dataset contains 285,173 geo-tagged tweets in total. To accommodate the inaccuracy of GPS, we assume an object to be located anywhere in a 10 meter radius of the point reported by the Twitter FireHose.

In the following, we conduct three sets of experiments and discuss the results. First, we test how CLUSTO configuration parameters, i.e., $minQ$ and $minSize$, affect clustering quality. This enables us to choose an optimal configuration for the subsequent experiments. Second, we measure the runtime of CLUSTO. Third, we compare our approach against DBSCAN.

### B. Varying $minQ$ and $minSize$

In the first set of experiments, we study how varying $minQ$ and $minSize$ impact clustering in CLUSTO. Intuitively, the closer $minQ$ is to zero, the more clusters are formed. However, the formed clusters are of poor quality because more noise objects are allowed to be in each cluster. On the other hand, the closer $minQ$ is to 1, the higher the quality is of the clusters that are created. However, setting it too high can result in missing valid clusters. Choosing a value for $minSize$ involves a similar trade-off.

*1) Cluster Coverage:* We start with evaluating the coverage of the discovered clusters. We make use of the Google Places API as follows. Initially, a random object is fetched from Google Places within the region considered. Google Places may return road names or other objects that are not considered

[2]http://developers.google.com/places/

as points of interest. To prevent these type of places, we require objects to have one or more ratings[3].

Then, we search among the clusters of CLUSTO to see if a similarly annotated cluster exists. Google Places objects have specific coordinates, while ours may span a larger region. To accommodate for imprecision in Google Places or in user locations, we allow a search radius of 25 meters from the Google Places object. We report a match if a cluster exists within the search radius that has an annotation that contains one or more similar terms. For example, if a randomly selected object from Google Places has the name "Hotel Tomo" and CLUSTO detected a cluster with either of the annotations "hotel," "tomo," or "hotel tomo," it is considered a match. We remove all stop words before the matching.

For each clustering, we fetch 100 random places from Google Places and perform the above matching. Figure 7 shows the results. As expected, more matches are found when more noise is allowed, i.e., with $minQ$ values closer to 0. However, a weaker requirement implies that false clusters might be created. The matching decreases from almost 80% to less than 10% when increasing $minQ$ from 0.1 to 0.9. This includes a sharp decrease when $minQ > 0.5$. In contrast, varying $minSize$ has just a slight impact on the matching.

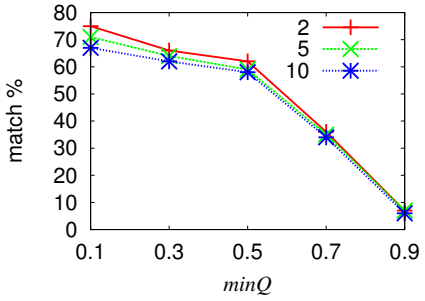



Fig. 7: CLUSTO Matches of Google Places

To show whether CLUSTO discovered clusters exist in Google Places, we perform an "opposite" matching, too. That is, we randomly select 100 CLUSTO clusters and search (using their annotations) for analogous PoIs in Google Places. The same setting (search radius, term matching) as above is used for the matching. The results are shown in Figure 8 (the value of $minQ = 0.9$ is omitted as not enough clusters are formed for all settings). In contrast to the previous experiment, the impact of $minQ$ is smaller, while the impact of $minSize$ is bigger. This indicates that setting $minSize$ too low may result in more false clusters. CLUSTO achieves a relatively high and stable 80–85% matching unless very poor clusters are allowed. That is, only with $minQ = 0.1$ and $minSize = 2$, the matching drops below 65%.

In the following experiments, we fix the value of $minSize$ to 5 since it demonstrates good results for both experiments.

[3]We verified that this procedure yields a PoI in the region.

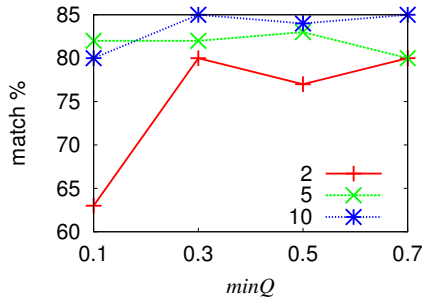



Fig. 8: Google Places Matches of CLUSTO Clusters

*2) Cluster Properties:* In this experiment, we consider how the characteristics of the discovered clusters change when $minQ$ is varied. We ran the experiment on the same dataset (containing 285,173 tweets) and analyzed the formed clusters with the following five popular annotations: "hotel," "food," "park," "coffee," and "restaurant." The number of posts mentioning the popular annotations are given in Table II.

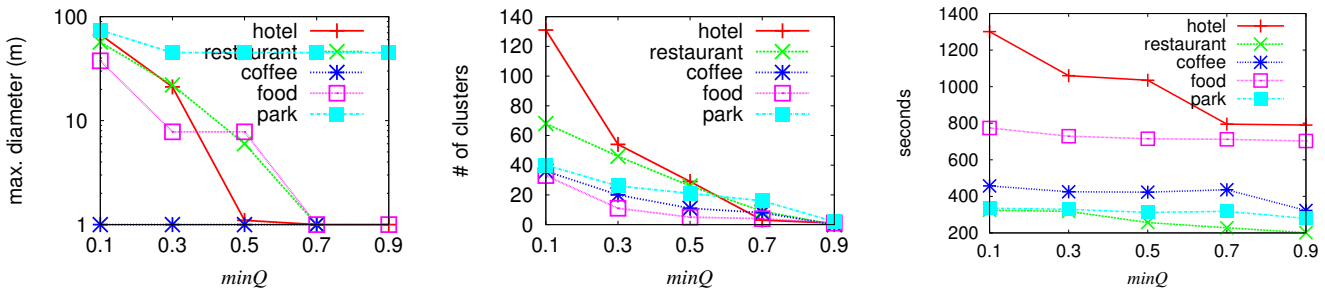
| *hotel* | *food* | *park* | *coffee* | *restaurant* |
|---|---|---|---|---|
| 5273 | 3591 | 2641 | 2484 | 2306 |

TABLE II: Number of Objects in the Dataset

First, we examine the changes in the spatial extent of the clusters by measuring their maximum diameter. Recall that a too low value of $minQ$ can result in over-expanding the clusters. For example, if two "hotels" are located within close proximity, they can be merged into one. Therefore, while we still can have a match of "hotel" in Google Places, the clustering does not provide accurate information about the region. Figure 9a shows the results. As expected, the diameters of the clusters decrease when $minQ$ approaches 1. However, for the clusters that are expected to be larger (i.e., "park"), CLUSTO is able to perform relatively stable (the diameter remains large) also with higher values of $minQ$. A cluster with the maximum diameter of 1 meter corresponds to a precisely located cluster (recall that we assume a GPS inaccuracy of 10 meters).

Next, Figure 9b shows the total number of clusters formed with each of the five annotations. The numbers decrease as expected for all annotations as $minQ$ approaches 1, due to the stronger requirement for forming clusters. We notice that for values close to 0, more clusters are created in close proximity to clusters with similar annotations. For example, a cluster with annotation "park" may have a cluster with the same annotation in close proximity, separated by a number of noise objects. This is the result of the weaker requirement that may introduce more clusters, but may not always be able to merge the clusters when too many noise objects exist.

As seen from Figure 9a and 9b, a fixed value of $minQ$ is suitable for different annotations. The diameter and number of clusters formed generally decreases for all annotations when $minQ$ approaches 1.




(a) Sizes of the Clusters (b) Number of Clusters (c) Runtime of CLUSTO

Fig. 9: Cluster Properties

### C. Runtime

We continue with the five popular annotations and measure the runtime of each clustering. CLUSTO runs offline and only once for a dataset, making the runtime less critical. The results are shown in Figure 9c. The runtime of a particular clustering depends on the number of candidate objects and roughly follows the order given in Table II. As such, "hotel" with most objects (5,273) takes the longest to process. Although "park" has slightly more objects than "coffee," clustering the former is faster than clustering the latter. We notice that this is because "park" objects are less spread in the region and are closer to each other than are "coffee" objects. Also, the running time tends to decrease for all annotations when $minQ$ increases. This is because the higher $minQ$ is set, the earlier the stopping condition is triggered in each clustering. This is especially true for the numerous "hotel."

### D. Annotation Properties

The proposed solution prioritizes the processing of longer annotations for two reasons. First, it makes the entire clustering run faster because each call to Algorithm 2 operates on smaller candidate sets. That is, the number of objects sharing the same longer (more specific) annotation (e.g., "Central Park") are likely to be smaller than the number of objects sharing the same shorter (more general) annotation (e.g., "Park"). As such, the processing of longer annotations is faster. Also, since the objects that successfully form a cluster are not reused, the candidate sets for shorter annotations shrink, making their processing faster as well. Second, longer annotations are likely to describe PoIs better. CLUSTO is therefore able to report relevant PoIs to the user as soon as it starts running.

In this experiment, we vary $minQ$ and count the number of terms in each annotation forming a cluster. The results are shown in Figure 10. When $minQ$ is set to 0.1, the majority of the annotations have 1 or 2 terms. This is because the low threshold allows more noise, and it is more likely that users share single terms rather than multiple. When $minQ$ increases to 0.3, 0.5, or 0.7, the number of terms in each annotation increases. Thus, single-term clusters are more likely to contain noise and thus are rejected by the stronger requirement. In general, the results confirm that CLUSTO is able to detect multi-term annotated clusters representing more descriptive PoIs. The results for $minQ = 0.9$ are omitted because not enough clusters are formed.

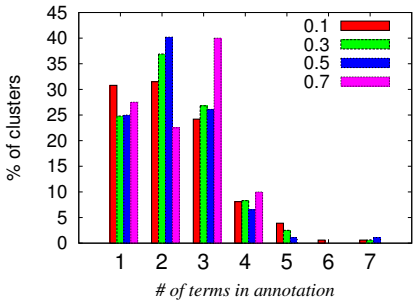



Fig. 10: Distribution of Number of Terms in Annotations

With the results so far, we propose a good value of $minQ$ for the current dataset. First, we want to avoid over-expansion, reducing noisy clusters, and to have the most descriptive annotations. Therefore, we avoid the value 0.1. Second, we want to maximize the number of clusters. Thus, we propose to use either 0.3 or 0.5. With a value of 0.3, we get more clusters and more matches with Google Places while preserving a reasonable size of the clusters.

### E. Popular Annotation Properties

In this experiment, we fix $minQ$ at 0.3 and report the number of terms each of the five popular annotations contains. Naturally, the number of terms vary greatly for each annotation as seen in Figure 11. The names of the places and how users mention them affect the clustering. For example, when the majority of the users mention "food" without any other common term, most of the resulting annotations have only this single term. The annotation "hotel" most often occurs with one additional term, and "restaurant" is most often combined with two additional terms.

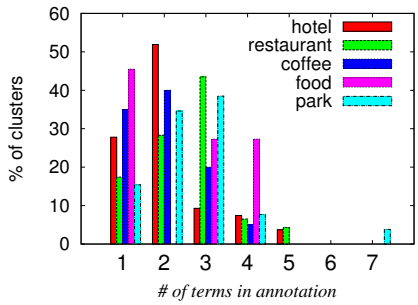



Fig. 11: Number of Terms in Popular Annotations

### F. CLUSTO versus DBSCAN

In the last set of experiments, we compare the clustering of CLUSTO with the clustering of DBSCAN [5]. In CLUSTO, we fix $minSize$ and $minQ$ to 5 and 0.3, respectively.

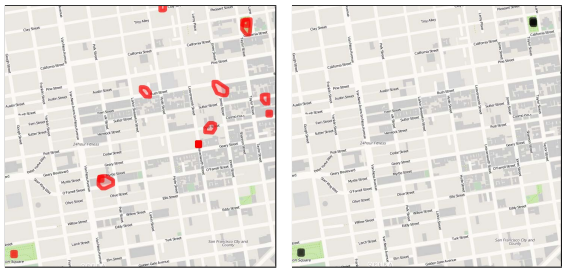
Similarly, DBSCAN requires a minimum number of points ($MinPts$) in a cluster, which we set to 5. Also, it requires a distance threshold parameter ($\epsilon$). Setting it too high may result in over-expanded clusters, while setting it too low may miss larger clusters. Based on the previous experiments, we know that cluster sizes vary and can reach 50 meters in diameter. To ensure the capability to form clusters with this diameter, the value of $\epsilon$ must be equally high. However, to avoid over-expansion, which we show in the following experiment, we choose a relatively low value of 25 meters, to make the comparison fair. Note that with this low value, DBSCAN is not be able to combine objects located more than 25 meters apart.

The resulting clustering for "hotel" is shown in Figure 12. It is clear from Figure 12a that DBSCAN over-expands clusters when the data is dense. As a result, the formed clusters do not correspond to the actual points of interest. CLUSTO, on the other hand, provides quite accurate clusters as seen from Figure 12b.

(a) DBSCAN, $\epsilon = 25$ m (b) CLUSTO, $minQ = 0.3$

Fig. 12: Clustering for "hotel"

DBSCAN does not take into account the textual attribute of other nearby objects, and it performs clustering based purely on spatial density. As a result, it may form clusters with annotations that are not dominating in the region. This effect is clearly visible in Figure 13a: DBSCAN finds "park" clusters where there are obviously no parks. CLUSTO takes into account the textual attribute and expands the clusters based on the merged quality function. Therefore, it forms clusters only with annotations that are dominating in the region. In Figure 13b, we see that no false "park" points of interests are formed.



(a) DBSCAN, $\epsilon$=25m (b) CLUSTO, $minQ$=0.3

Fig. 13: Clustering for "park"

Finally, we compare the maximum diameters of the clusters in Figure 14. We use the five popular annotations given in Table II. The diameters of the clusters formed by DBSCAN are up to two orders of magnitude larger than those of CLUSTO. While a 200 meter diameter for "park" clusters may be reasonable, a 1,000 meter diameter for clusters "hotel" and "food" are unrealistic. In contrast, CLUSTO provides meaningful diameters for all annotations.

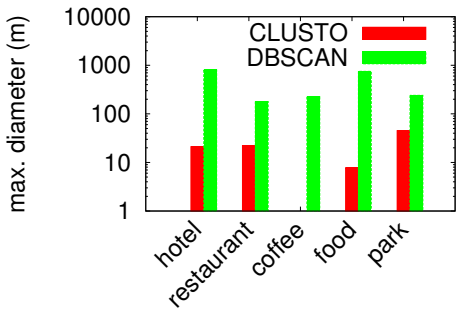



Fig. 14: Sizes of the Clusters

## V. Related Work

Recently, substantial research efforts have explored various research directions related to microblogs. This includes indexing of microblogs [1], [3], [21], [22], event detection from microblogs [12], [13], [16], news extraction from microblogs [18], microblog posts based recommendations [6] and decision making [2] systems, microblog ranking [4], [20], visualization [11], and spatio-temporal aggregation [19].

Despite such rich research on microblogs, to the best of our knowledge, no existing work that addresses the discovery of points of interests using geo-tagged microblog posts, which is the focus of this paper. However, the paper's proposal is closely related to spatial clustering.

**Spatial Clustering.** For decades, there has been extensive research on clustering algorithms and their applications in

many areas [8]. Our work is closely related to density-based partitioning algorithms [9]. In these, a set of data objects form a cluster if they are spread in the data space over a contiguous region of high object density. Density-based algorithms aim to identify all such dense regions that are separated by low-density regions. Data objects located in low-density regions are considered as noise. Well-known examples of such algorithms include DBSCAN [5], its extension GDBSCAN [17], and DENCLUE [7]. The major advantage of these algorithms over the classic partitioning approaches (e.g., $k$-means [10] or CLARANS [15]) are that they do not require the number of clusters as an input parameter ($k$), which would be a crucial limitations in our setting, too. To apply density-based clustering to our setting, i.e., to take into account the textual attribute, the algorithm has to be run for each text annotation separately. Thus, density-based clustering does not consider objects with other text annotations, which may result in over-expansion of clusters. Further, density-based is limited by the search range, which on the other hand may result in under expansion of clusters. Both cases are evaluated in Section IV-F.

In CLUSTO, the proposed quality function takes into account all nearby objects, and an object is added to a cluster only if their merged quality is not below the given threshold. Although similar to a minimum density (defined by $\epsilon$ and $MinPts$ in DBSCAN), our approach, roughly speaking, expands a cluster as long as the annotation is dominating in the region. Thus, clusters of varying density can be formed, which is not possible with density-based clustering.

To choose the next candidate for a cluster, CLUSTO borrows techniques from agglomerative clustering based on nearest neighbor (NN) chains [14]. An NN-chain consists of an arbitrary cluster, followed by its NN, which is again followed by its NN from among the remaining clusters, and so on. Such an NN-chain ends in a mutual or reciprocal NN (RNN) pair, i.e., a pair of clusters $c_1$ and $c_2$ such that the NN of $c_1$ is $c_2$, and vice versa. We found RNN clustering particularly efficient in our setting for two reasons. First, the merged RNN pair does not affect the remaining chain members, and thus can be reused for the subsequent agglomerations. Second, the spatial NN search can be accelerated using spatial indexing.

## VI. Conclusions and Future Work

We present a spatio-textual clustering method for the discovery of points of interest from geo-tagged microblog posts. The method takes into account both spatial proximity and textual relevance and is able to form clusters of arbitrary shape and density. A proposed merged cluster quality function serves as a criterion for cluster expansion and, in combination with nearest-neighbor chaining, prevents over-expansion of clusters. An experimental study with real data offers insight into the properties of the resulting clusters; and it demonstrates that the method is able to extract accurate and comprehensive PoIs in a realistic, real-world setting.

Since all microblog posts are timestamped, interesting future work includes adding a temporal dimension to the spatio-textual clustering. This may enrich the points of interest further, e.g., by labeling them with inferred opening hours. Also, in the proposed solution, we perform simple preprocessing of the textual description of objects. This may be extended using more advanced natural-language processing techniques.

**Acknowledgments** We thank the reviewers for their helpful comments. The research was supported in part by the Danish National Research Foundation grant DNRF84 through Center for Massive Data Algorithmics (MADALGO) and by a grant from the Obel Family Foundation.

## References

[1] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-time search at twitter. In *ICDE*, pages 1360–1369, 2012.
[2] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? Jury selection for decision making tasks on micro-blog services. *Proc. VLDB Endowment*, 5(11):1495–1506, 2012.
[3] C. Chen, F. Li, B. C. Ooi, and S. Wu. TI: an efficient indexing mechanism for real-time search on tweets. In *SIGMOD*, pages 649–660, 2011.
[4] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: improving recency ranking using twitter data. In *WWW*, pages 331–340, 2010.
[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.
[6] J. Hannon, M. Bennett, and B. Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *RecSys*, pages 199–206. ACM, 2010.
[7] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *KDD*, volume 98, pages 58–65, 1998.
[8] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990.
[9] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *WIREs Data Mining Knowl Discov*, 1(3):231–240, 2011.
[10] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proc. Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.
[11] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. Processing and visualizing the data in tweets. *SIGMOD Rec.*, 40(4):21–27, 2012.
[12] M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. *Proc. VLDB Endowment*, 3(1-2):1091–1102, 2010.
[13] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, pages 1155–1158, 2010.
[14] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.
[15] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.
[16] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW*, pages 851–860, 2010.
[17] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.
[18] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *GIS*, pages 42–51, 2009.
[19] A. Skovsgaard, D. Sidlauskas, and C. S. Jensen. Scalable top-k spatio-temporal term querying. In *ICDE*, pages 148–159, 2014.
[20] I. Uysal and W. B. Croft. User oriented tweet ranking: a filtering approach to microblogs. In *CIKM*, pages 2261–2264, 2011.
[21] X. Xiao, Y. Xu, L. Wu, and W. Lin. LSII: An indexing structure for exact real-time search on microblogs. In *ICDE*, pages 482–493, 2013.
[22] J. Yao, B. Cui, Z. Xue, and Q. Liu. Provenance-based indexing support in micro-blog platforms. In *ICDE*, pages 558–569, 2012.