Effective Location Identification from Microblogs

Guoliang Li[†] Jun Hu[†] Kian-lee Tan[‡] Zhifeng Bao[‡] Jianhua Feng[†]
[†]*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.*
[‡]*School of Computing, National University of Singapore, Singapore.*
{liguoliang,fengjh}@tsinghua.edu.cn; hu-j12@mails.thu.edu.cn; {tankl,baozhife}@comp.nus.edu.sg

***Abstract*—The rapid development of social networks has resulted in a proliferation of user-generated content (UGC). The UGC data, when properly analyzed, can be beneficial to many applications. For example, identifying a user's locations from microblogs is very important for effective location-based advertisement and recommendation. In this paper, we study the problem of identifying a user's locations from microblogs. This problem is rather challenging because the location information in a microblog is *incomplete* and we cannot get an accurate location from a *local* microblog. To address this challenge, we propose a *global* location identification method, called GLITTER. GLITTER combines multiple microblogs of a user and utilizes them to identify the user's locations. GLITTER not only improves the quality of identifying a user's location but also supplements the location of a microblog so as to obtain an accurate location of a microblog. To facilitate location identification, GLITTER organizes points of interest (POIs) into a tree structure where leaf nodes are POIs and non-leaf nodes are segments of POIs, e.g., countries, states, cities, districts, and streets. Using the tree structure, GLITTER first extracts candidate locations from each microblog of a user which correspond to some tree nodes. Then GLITTER aggregates these candidate locations and identifies top-k locations of the user. Using the identified top-k user locations, GLITTER refines the candidate locations and computes top-k locations of each microblog. To achieve high recall, we enable fuzzy matching between locations and microblogs. We propose an incremental algorithm to support dynamic updates of microblogs. Experimental results on real-world datasets show that our method achieves high quality and good performance, and scales very well.**

I. INTRODUCTION

With the rapid development of social networks, the amount of user generated content (UGC) is increasing at an alarming rate. For example, Twitter has 140 million active users and generates 400 million tweets per day. Foursquare has over 25 million users and 3 billion check-ins. Many applications can benefit from the UGC data. Specifically, extracting locations from microblogs can result in more effective location-based advertisement and recommendation. For example, by identifying the location with respect to a user's microblog (e.g., "`Olympia Theater`, `Broadway`, `Manhattan`"), advertisers can send relevant advertisements to the user as soon as the user posts the microblog. More importantly, based on all microblogs posted by the user, if we can get the interested location of the user (e.g., "`Manhattan`"), we can provide localized (news, products, restaurants) recommendation.

In this paper, we study the problem of identifying locations from microblogs, including microblog location and user location. Microblog location is fine-grained and may be the current location of the user, e.g., shopping mall, restaurant, and scenic spot. User location, on the other hand, is coarse-grained and is usually a small region that is frequently mentioned by the user in multiple microblogs, e.g., home location and office location. However it is a challenging problem to identify such locations. First, few people (16%) register accurate locations in their profile; moreover, many users either leave out their location information or provide locations that are not useful (e.g., "my home") [17]. Second, location information of a microblog is *incomplete* and it is rather hard to obtain an accurate location from a *local* microblog. For example, from a microblog "`Olympia Theater` is so nice," we are unable to identify the accurate location of the microblog as there are many "`Olympia Theaters`" across the world. However if we know the user also posts a microblog "`Manhattan` is my favorite place," we can infer that the "`Olympia Theater`" mentioned in the earlier microblog is very likely (with high probability) to be in "`Manhattan`, `New York`."

To overcome these challenges, we propose GLITTER, a *global* location identification method, to infer accurate locations from microblogs. GLITTER combines multiple microblogs of a user and utilizes them to effectively identify locations from microblogs. GLITTER not only identifies a user's interested locations but also supplements the location of a microblog in order to obtain its location accurately.

To describe location information, GLITTER organizes points of interest (POIs) into a tree structure where the leaf nodes are POIs and non-leaf nodes are segments of POIs, e.g., countries, states, cities, districts, and streets. Using the tree structure, GLITTER first extracts candidate locations from each microblog of a user which correspond to some tree nodes. Then GLITTER aggregates these candidate locations and identifies top-k locations of the user. Using the identified top-k locations, GLITTER refines the candidate locations and computes top-k locations of every microblog of the user. To achieve high recall, we enable fuzzy matching between microblogs and locations (e.g., "`Mahatan`" in a microblog can match location "`Manhattan`"). We also propose an incremental algorithm to efficiently support dynamic updates of microblogs. To summarize, we make the following contributions.

- We propose a global location identification method for inferring top-k locations of a user from her microblogs and top-k locations for each of her microblogs.
- We devise a three step framework to address this problem. The extraction step extracts candidate locations from microblogs. The aggregation step aggregates candidate locations and generates top-k locations of the user. The refinement step refines candidate locations and computes top-k locations of each microblog.
- We enable fuzzy matching between locations and microblogs to achieve high recall. We develop an incremental algorithm to support dynamic updates of microblogs.
- We have implemented our method and experimental results on real-world datasets show that our method achieves high quality and good performance, and scales very well.

Paper Structure: We first formulate the problem in Section II and then review related works in Section III. Our GLITTER framework is presented in Section IV. We introduce the extraction model, aggregation model, and refinement model in Sections V, VI, and VII respectively. Section VIII gives an incremental algorithm to support updates. Experimental results are reported in Section IX. We conclude in Section X.

II. PROBLEM FORMULATION

Consider a microblog system, e.g., Twitter, where a user posts a set of microblogs $M = \{m_1, m_2, \ldots, m_{|M|}\}$. Each microblog m_i consists of a set of tokens. Given a set of POIs $P = \{p_1, p_2, \ldots, p_{|P|}\}$ [1], where each POI includes a name and a location, we focus on utilizing POI names and locations to identify the top-k locations of each microblog and the top-k locations of the user. For example, Table I shows 12 microblogs and 12 POIs. The top-1 location of m_3="I was able to get a tour at `Film School`, `Sunset blvd`" should be "`Sunset Blvd`, `Hollywood`, `Los Angeles`, `California`" based on POI p_3 with name of "`Film School`". The top-2 locations of the user should be "`Hollywood`, `Los Angeles`, `California`" ("`Hollywood`" for short) and "`San Diego`, `California`" ("`San Diego`" for short).

Challenges. To identify the locations of a microblog, a naive method identifies the most similar POIs that have the largest similarity to the microblog using extraction based techniques [5], [14], [10]. For example we can use the Jaccard coefficient to quantify the similarity between a POI and a microblog, which is the ratio of their overlap size to their union size. However this method has some limitations. First, a single microblog may be incomplete to extract the accurate location. For example, we cannot identify the location accurately from the microblog "I am at `Film School`", because there are many "`Film Schools`" and we cannot determine the exact location based on the limited text. Second, we cannot identify the location of a user since the user location may not be a POI and should be a residential zone or a commercial district. For example, the locations of a user should be "`Hollywood`" or "`San Diego`". To address the first challenge, we propose a global location identification method for extracting locations from microblogs, called GLITTER. GLITTER utilizes multiple microblogs of a user to identify top-k locations for the user and top-k locations for each microblog (see Section IV). To address the second challenge, we construct a tree-based location structure to organize the POIs as follows.

Tree-based Location Structure. We first obtain a location hierarchy from existing knowledge bases, e.g., Yago [12][2], which includes location concepts: country, state, city, district, and street. Then based on the location hierarchy, we segment the location of each POI into: country, state, city, district, and street. We call each segment a *location entity*. We can construct a tree-based location structure using these entities. Each tree node is labelled with an entity. The first level (root) is the whole space with empty label. The second level consists of country entities. The third level includes state entities. The fourth level consists of city entities. The fifth level includes district entities. The sixth level consists of street entities. The seventh level (leaf) includes all POI names.

[1]We can get POIs in the world from many sources, e.g., factual.com.

[2]Available at http://www.mpi-inf.mpg.de/yago-naga/yago/

Notice that each node corresponds to a location consisting of location entities of nodes on the path from the root to the node. Each path from the root to a leaf node corresponds to a POI (Point Location) and each path from the root to a non-leaf node corresponds to a region (Range Location). To facilitate location identification, we assign each node with a Dewey code in a top-down manner. The Dewey code of the root is 1. The Dewey code of a node is computed by appending its sequence number among its siblings to its parent's Dewey code. Based on the Dewey code, we can easily get the ancestor-descendant relationship between two nodes: considering two nodes n_i and n_j, if n_i's Dewey code is a substring of n_j's Dewey code, then n_i is an ancestor of n_j, and vice versa.

Figure 1 shows the tree-based location structure constructed from the POIs in Table I (Here we only show the subtree under the `California` entity). Consider POI p_3 with location "`Sunset Blvd`, `Hollywood`, `Los Angeles`, `California`". We segment it into four location entities "`California`", "`Los Angeles`", "`Hollywood`", and "`Sunset Blvd`". Their corresponding nodes are respectively 1, 1.1, 1.1.1, and 1.1.1.2. Node 1.1 is an ancestor of node 1.1.1.2. Node 1.1.1 denotes the address of "`Hollywood`, `Los Angeles`, `California`".

Problem Formulation. Given a set of POIs P, let T denote the tree-based location structure constructed by POIs in P. Given a user with a set of microblogs M, for each microblog in M, we identify k best tree nodes from T as its top-k locations. For the user, we identify k best tree nodes from T as her top-k locations. Notice that the microblog locations would be leaf nodes and user locations would be non-leaf nodes.

For example, consider the microblogs in Table I and the location-based tree structure in Figure 1. The top-1 location of microblog m_3 should be "`Film School`, `Sunset Blvd`, `Hollywood`, `Los Angeles`, `California`". The top-2 locations of the user should be "`Hollywood`" and "`San Diego`".

III. RELATED WORK

Location Identification. The work most related to ours is to estimate a user's city-level location from microblogs. Cheng et al. [9] proposed a probabilistic framework for estimating a Twitter user's city-level location. However their methods cannot identify locations at different levels (e.g., states, cities, districts). Chandraet al. [7] improved this probabilistic model by utilizing the re-tweet information. Amitay et al. [2] extracted geo scope from web pages using heuristics rules. Backstrom et al. [3] proposed a framework for modeling the spatial variation in search queries. They used search-engine query logs and geolocation techniques to assign accurate locations to the IP addresses issuing the queries. Unlike our method, these approaches focus on identifying users' city-level locations and cannot extract the accurate location of a microblog. Moreover, they cannot support fuzzy matching between locations and microblogs.

Entity Extraction. There are many studies on dictionary-based entity extraction [6], [14], [21], [10], [18], [23], [8], [1], [5], where, given a dictionary of entities and a document, extracts all substrings from the document that match some entities in the dictionary. Chandel et al. [6] proposed a batch based method to support efficient extraction by utilizing indexed entities. Chakrabarti et al. [5] enabled fuzzy matching between the entity and the substring, which uses similarity functions to quantify the similarity between an entity and a substring,

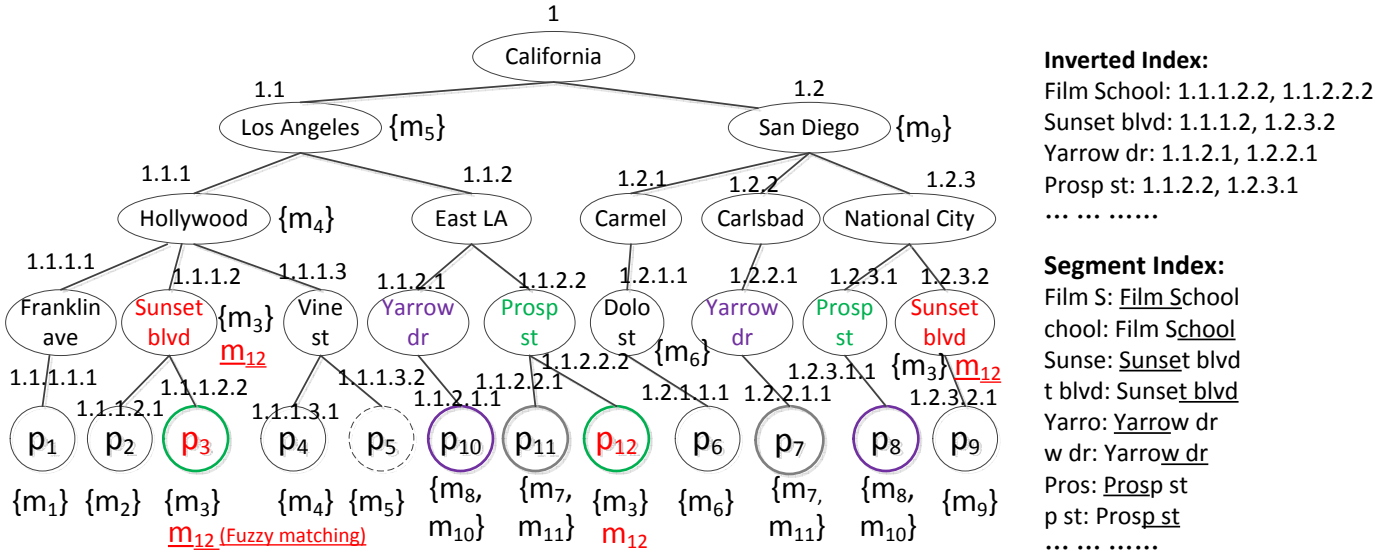
TABLE I. EXAMPLE MICROBLOGS AND POIS.

(a) Microblogs

ID	Microblogs
m_1	Nice. @Candilejas Night Club.
m_2	Muffin bar is so nice. @Highland Gardens Hotel.
m_3	I was able to get a tour at Film School, Sunset blvd.
m_4	I feel much better @Health Clinic, Hollywood.
m_5	My favorite coffee in Los Angeles. @Groundwork Coffee
m_6	Finding jobs from Worksource Center in Dolo St.
m_7	Just passed by the Life Point Counseling Center.
m_8	Bought shoes from Sports Shop Center.
m_9	Perfect Gift Store - Victoria's Gift Shop in San Diego.
m_{10}	Sports Shop Center is good for shopping.
m_{11}	Get Lost. @Counseling Center.
m_{12}	FilmSchool @Sunsat blvd is so nice.

(b) POIs

ID	POI Name	Location
p_1	Candilejas Night Club	Franklin ave, Hollywood, Los Angeles, California
p_2	Highland Gardens Hotel	Subset blvd, Hollywood, Los Angeles, California
p_3	Film School	Subset blvd, Hollywood, Los Angeles, California
p_4	Health Clinic	Vine st, Hollywood, Los Angeles, California
p_5	Groundwork Coffee	Vine st, Hollywood, Los Angeles, California
p_6	Worksource Center	Dolo st, Carmel, San Diego, California
p_7	Counseling Center	Yarrow dr, Carlsbad, San Diego, California
p_8	Sports Shop Center	Prosp st, National City, San Diego, California
p_9	Victoria's Gift Shop	Subset blvd, National City, San Diego, California
p_{10}	Sports Shop Center	Yarrow dr, East LA, Los Angeles, California
p_{11}	Counseling Center	Prosp st, East LA, Los Angeles, California
p_{12}	Film School	Prosp st, East LA, Los Angeles, California



Inverted Index:

Film School: 1.1.1.2.2, 1.1.2.2.2
Sunset blvd: 1.1.1.2, 1.2.3.2
Yarrow dr: 1.1.2.1, 1.2.2.1
Prosp st: 1.1.2.2, 1.2.3.1
… … ……

Segment Index:

Film S: Film School
chool: Film School
Sunse: Sunset blvd
t blvd: Sunset blvd
Yarro: Yarrow dr
w dr: Yarrow dr
Pros: Prosp st
p st: Prosp st
… … ……

Fig. 1. Tree-based Location Structure and Indexes.

e.g., Jaccard coefficient, Cosine, and Edit distance. Wang et al. [23] proposed neighbor enumeration based method to support the edit-distance function. Li et al. [14] proposed a unified method to support various similarity functions. Deng et al. [10] proposed a segment based method to improve the extraction performance. These methods are effective for microblogs with complete locations (e.g., a microblog with the whole location of a POI). However they are rather ineffective for microblogs with incomplete locations because these microblogs contain only partial keywords of locations and cannot match to any locations. To address these limitations, we propose a global extraction based method.

Other Related Works. There are some machine learning based techniques for mining location-based information. Mei et al. [19] proposed a topic model to mine spatiotemporal themes from web logs. Rattenbury et al. [20] proposed a generalizable approach for extracting tag semantics (e.g., event and place) based on the distribution of Flickr tags. Backstrom et al. [4] studied the relationship between friendship and spatial distance and predicted a user's location based on their friends. Li et al. [17] improved this model by using a unified discriminative influence model. Hao et al. [11] introduced Gibbs sampling into the location based topic modelling. Yin et al. [25] studied the problem of discovering and comparing geographical topics from GPS-associated documents. Hong et al. [13] proposed a geographical topical model by utilizing both statistical topic models and sparse coding techniques.

IV. THE GLITTER FRAMEWORK

In this section, we introduce our global location identification framework, GLITTER, which includes three steps.

- **Extraction Step:** This step extracts *candidate locations* of each microblog which are substrings of the microblog that exactly (or approximately) match location entities on some tree nodes. We will formally define the concept of candidate locations in Section V. There are two main challenges in extracting the candidate locations. The first one is how to evaluate the proximity between a microblog and a tree node. The second one is how to efficiently extract the tree nodes that are highly relevant to the microblog. We propose a similarity based model and devise efficient extraction algorithms in Section V.
- **Aggregation Step:** This step aggregates candidate locations of every microblog and generates top-k user locations based on these candidates. One big challenge is how to define the proximity between a tree node and the user. We propose an effective aggregation model and devise efficient algorithms to address this challenge in Section VI.
- **Refinement Step:** This step refines candidate locations of each microblog using the identified top-k user locations and generates top-k locations of each microblog. We present a refinement model to identify the top-k locations for each microblog in Section VII.

The pseudo-code of our algorithm is illustrated in Figure 2. GLITTER extracts candidate locations for each microblog by calling function GLITTER-EXTRACTION (line 4), which will be discussed in Section V. Then it aggregates these candidate locations to compute top-k locations of the user by calling function GLITTER-AGGREGATION (line 6), which will be presented in Section VI. Finally based on top-k user locations, it refines the candidate locations for each microblog by calling

Algorithm 1: GLITTER (M, k, T)

Input: $M = \{m_1, m_2, \cdots, m_{|M|}\}$: Microblogs of a user; k: An integer; T : Location structure
Output: $\mathcal{R}_u$: The *top-k* locations of the user; $\mathcal{R}_{m_i}$: The *top-k* locations of microblog m_i

```
1 begin
2     L = φ ;
3     foreach microblog m_i do
4         N(m_i) = GLITTER-EXTRACTION (m_i, T) ;
5         L ← N(m_i) ;
6     R_u = GLITTER-AGGREGATION (L, k, T) ;
7     foreach microblog m_i do
8         R_{m_i} = GLITTER-REFINE (R_u, k, m_i, T) ;
9 end
```

Fig. 2. GLITTER Algorithm.

function GLITTER-REFINE (line 8), which will be discussed in Section VII. Notice that we can iteratively call functions GLITTER-AGGREGATION and GLITTER-REFINE multiple times until the *top-k* locations for the user are stable.

For example, consider the microblogs in Table I and tree-based location structure in Figure 1. In the extraction step, for each microblog, e.g., m_3, we extract location entities: "`Film School`" and "`Sunset blvd`". Based on the two entities, we get four tree nodes: 1.1.1.2, 1.1.1.2.2, 1.1.2.2.2, 1.2.3.2. We highlight the extracted tree nodes of every microblog in the figure. In the aggregation step, we aggregate all of these tree nodes and compute top-2 locations of the user: nodes 1.1.1 and 1.2. In the refinement step, using the top-2 user locations, we refine the location of each microblog. As many of the user's microblogs are relevant to node 1.1.1, the top-1 location of m_3 should be location 1.1.1.2.2.

V. LOCATION EXTRACTION

This section studies how to extract the candidate locations of each microblog. We first propose an exact extraction method in Section V-A and then extend it to support fuzzy matching between microblogs and location entities in Section V-B.

A. Exact Extraction

We discuss how to extract substrings from a microblog that exactly match some location entities. Before we introduce the details, we first introduce two concepts.

Definition 1 (Exact-matching Entities): Given a microblog and a tree-based location structure, a location entity that exactly matches a substring of the microblog is called an exact-matching entity.

Definition 2 (Exact-matching Tree Nodes): The tree nodes whose labels are exact-matching entities are called exact-matching tree nodes.

Since the entities of exact-matching tree nodes appear in the microblog, these tree nodes should be relevant to the microblog and thus we take them as *candidate locations*. For instance, consider microblog m_3="I was able to get a tour at `Film School`, `Sunset blvd`". "`Film School`" and "`Sunset blvd`" are two exact-matching entities and 1.1.1.2.2, 1.1.2.2.2, 1.1.1.2, 1.2.3.2 are exact-matching tree nodes.

Algorithm 2: GLITTER-EXTRACTION (m_i, T)

Input: m_i: A microblog; T : Location structure
Output: $N(m_i)$: The candidate locations;

```
1 begin
2     Extracting exact/fuzzy matching entities;
3     Identifying exact/fuzzy matching nodes;
4     Computing similarities of candidate locations;
5     Add candidate locations and similarities to N(m_i);
6 end
```

Fig. 3. GLITTER-EXTRACTION Algorithm(Extracting Candidate Locations).

Next we discuss how to efficiently identify exact-matching tree nodes and assign a weight to each node for measuring the proximity between the node and the microblog.

Location Extraction. To facilitate the exact extraction, we build an inverted index on top of the location entities. The entries of the inverted index are the location entities on the tree-based location structure. For each entity e, we maintain an inverted list of tree nodes whose corresponding entities are e. To identify location candidates, we enumerate the substrings of the microblog, and for each substring we check whether it appears on the inverted index. If yes, we get the tree nodes from the inverted list of this substring and these tree nodes are exactly candidate locations. (Notice that we can utilize batch based extraction techniques [6] to improve the performance.) For example, in Figure 1, we show the inverted lists of some entities, e.g., "`Film School`", "`Sunset blvd`". Based on the inverted lists of these entities, we can efficiently obtain the exact-matching tree nodes (i.e., candidate locations).

Location Ranking. Since an exact-matching tree node corresponds to a *candidate location*, the more number of entities on the path from the root to the tree node that appear in the microblog, this candidate location is more relevant to the microblog. In particular, if all entities on the path appear in the microblog, this candidate location must be very relevant to the microblog. For example, considering microblog m_3, nodes 1.1.1.2, 1.1.1.2.2, 1.1.2.2.2, 1.2.3.2 are exact-matching nodes. Node 1.1.1.2.2 is a better location than other nodes since it contains two entities while others contain only one entity.

Based on this observation, we devise a similarity function to quantify the similarity between a candidate location and a microblog. Given a microblog m, let $E(m)$ denote the set of exact-matching entities. Given a candidate location n, let $E(n)$ denote the set of entities of nodes on the path from the root to node n. We can utilize any set similarity functions on top of $E(m)$ and $E(n)$ to evaluate the similarity between microblog m and tree node n. In this paper we adopt the Jaccard coefficient as an example, which is defined as

$$\mathcal{J}_e(m, n) = \frac{|E(m) \cap E(n)|}{|E(m) \cup E(n)|}, \tag{1}$$

where $|E(m) \cap E(n)|(|E(m) \cup E(n)|)$ is the overlap (union) size of $E(m)$ and $E(n)$. Notice that our method can be easily extended to support other similarity functions, e.g., Cosine similarity and weighted Jaccard similarity.

Based on the similarity function, we can devise an extraction algorithm. Figure 3 shows the pseudo-code. It first extracts exact-matching entities. Then utilizing the inverted index, it identifies the corresponding exact-matching tree nodes. Next it computes the similarity between each node and the microblog.

Finally, it adds the tree nodes and the corresponding similarities into the candidate set.

For example, consider microblog m_3. We extract two exact-matching entities "Film School" and "Sunset blvd". $E(m_3)$ = {Film School, Sunset blvd}. From their inverted lists, we retrieve the exact-matching tree nodes. Consider exact-matching tree nodes {1.1.1.2.2, 1.1.2.2.2} in the inverted list of "Film School". $E(1.1.1.2.2)$ = {FilmSchool, Sunset Blvd, Hollywood, Los Angeles, California}. $\mathcal{J}_e(m_3, 1.1.1.2.2) = \frac{2}{5}$. $E(1.1.2.2.2)$ = {Film School, Prosp st, East LA, Los Angeles, California}. $\mathcal{J}_e(m_3, 1.1.2.2.2) = \frac{1}{6}$. Thus node 1.1.1.2.2 is more relevant to microblog m_3 than node 1.1.2.2.2. Similarly, for exact-matching tree nodes {1.1.1.2, 1.2.3.2} in the inverted list of "Sunset blvd", we have $\mathcal{J}_e(m_3, 1.1.1.2) = \frac{1}{5}$ and $\mathcal{J}_e(m_3, 1.2.3.2) = \frac{1}{5}$.

B. Fuzzy Extraction

Exact extraction cannot tolerate any inconsistencies between location entities and microblogs. For example, microblogs usually contain errors due to typing errors. To alleviate this problem, we enable fuzzy matching between microblogs and location entities.

Fuzzy Matching: To enable fuzzy matching, we utilize similarity functions to define the similarity between two entities. We take the well-known Levenshtein distance (also known as edit distance) as an example, and our method can be easily extended to support other functions, e.g., Jaccard coefficient. The edit distance between two entities is the minimum number of edit operations (including substitution, insertion, deletion) to transform one entity to another. We use the normalized edit similarity to make the similarity between 0 and 1. Given two entities e_1 and e_2, let $\texttt{ED}(e_1, e_2)$ denote their edit distance and $\texttt{EDS}(e_1, e_2) = 1 - \frac{\texttt{ED}(e_1,e_2)}{\max(|e_1|,|e_2|)}$ denote their normalized edit similarity. Two entities are similar if their normalized edit similarity is not smaller than a given threshold. For example, the edit distance between "sunsat blvd" and "sunset blvd" is 1 and their edit similarity is $\frac{10}{11}$. Based on the similarity function, we introduce two concepts.

Definition 3 (Fuzzy-matching Entities): Given a microblog, a tree-based location structure, a similarity function and a threshold, a location entity that *approximately matches* a substring of the microblog (i.e., the similarity between the location entity and the substring is not smaller than the given threshold) is called a fuzzy-matching entity.

Definition 4 (Fuzzy-matching Tree Nodes): The tree nodes whose labels are fuzzy-matching entities are called fuzzy-matching tree nodes.

For each microblog, in fuzzy extraction based method, we extract its fuzzy-matching tree nodes and take them as *candidate locations*. For example, consider microblog m_{12}. If we consider exact matching, we cannot extract any candidate location from the microblog. However, if we use fuzzy-matching based method, "FilmSchool" approximately matches location entity "Film School" and "Sunsat blvd" approximately matches location entity "Sunset blvd". Thus in fuzzy-matching based method, we can identify two fuzzy-matching entities from the microblog. From their inverted lists, we can obtain four fuzzy-matching tree nodes: 1.1.1.2, 1.1.1.2.2, 1.1.2.2.2, 1.2.3.2.

Location Extraction. To support efficient fuzzy extraction, we adopt a segment-based method [16], [15][3]. For simplicity, suppose two entities are similar if their edit distance is not larger than threshold τ. To find fuzzy-matching entities, we partition each entity into $\tau + 1$ segments. Based on the pigeonhole principle, we can prove that if a substring of a microblog is similar to an entity, the substring must contain a segment of the entity. Thus we can transform the fuzzy matching problem into the exact matching problem. To this end, we build a segment-based inverted index. The entries are segments. Each segment has an inverted list of entities that contain the segment. Then given a microblog, we enumerate its substrings and check whether they appear in the inverted index. If a substring appears in the inverted index, it matches a segment. Next we check whether each entity in the inverted list of the segment is similar to the substring. If yes, the entity is a fuzz-matching entity. Finally, from the inverted lists of fuzzy-matching entities, we identify fuzzy-matching tree nodes.

For example, Figure 1 shows the segment-based index. Suppose the edit distance threshold is 1. We partition each location entity into 2 segments. For entity "Film School", we partition it into "Film S" and "chool". Consider the substring "FilmSchool" of microblog m_{12}. We can find a substring "chool" which matches a segment. Thus we can identify fuzzy-matching entity "Film School" for "FilmSchool". From the inverted list of "Film School", we can get the fuzzy-matching tree nodes: 1.1.1.2.2, 1.1.2.2.2. For microblog m_{12}, we have two fuzzy-matching entities "Film School" and "Sunset blvd" and obtain four fuzzy-matching tree nodes: 1.1.1.2, 1.1.1.2.2, 1.1.2.2.2, 1.2.3.2.

Location Ranking. We extend Equation 1 to support fuzzy matching. Given a substring in the microblog, it may be similar to multiple entities, and we consider the most similar one. Similarly, given an entity, it may have multiple similar substings, and we also keep the most similar one. Let $E'(m_i)$ denote the set of pairs $\langle s, e\rangle$ where s is a substring and e is an entity, and there does not exist (1) $\langle s, e'\rangle$ such that $\texttt{EDS}(s, e') > \texttt{EDS}(s, e)$ and (2) $\langle s', e\rangle$ such that $\texttt{EDS}(s', e) > \texttt{EDS}(s, e)$. Then we use the following function to quantify the similarity between a microblog m and a location n,

$$\mathcal{J}_f(m, n) = \frac{\sum_{\langle s,e\rangle \in E'(m) \bowtie_e E(n)} \texttt{EDS}(s, e)}{|\pi_e(E'(m)) \cup E(n)|}, \qquad (2)$$

where $E'(m) \bowtie_e E(n)$ denotes the join result of $E'(m)$ and $E(n)$ on entities, and $\pi_e(E'(m))$ is the selection on entities.

For example, consider m_{12} = {FilmSchool@ Sunsat blvd is so nice}. We extract two fuzzy-matching entities "Film School" and "Sunset blvd" which are respectively similar to substrings "FilmSchool" and "Sunsat blvd". $E'(m_{12})$ = {⟨FilmSchool, Film School⟩, ⟨Sunsat blvd, Sunset blvd⟩}. $E(1.1.1.2.2)$ = {Film School, Sunset Blvd, Hollywood, Los Angeles, California}. Thus $\mathcal{J}_f(m_{12}, 1.1.1.2.2) = \frac{\frac{11}{12} + \frac{10}{11}}{5}$.

VI. Location Aggregation

Given a user, based on the candidate locations of each of her microblogs, we study how to integrate these candidate locations and identify top-k locations of the user. We first

[3]We can also use other indexes, e.g., trie [22].

discuss how to identify top-k locations from the same level in Section VI-A. Then we extend it to identify top-k locations from different levels in Section VI-B.

A. Identifying Top-k Locations from The Same Level

Consider a set of microblogs $M = \{m_1, m_2, \ldots, m_{|M|}\}$ posted by a user. Each microblog m_i is associated with a set of candidate locations (exact-matching or fuzzy-matching tree nodes), denoted by $N(m_i)$, which is identified in the extraction step. Each candidate location in $N(m_i)$ has a similarity with m_i. We want to use these candidate locations and the corresponding similarities to evaluate the relevance between any tree node n and the user location. Obviously node n is relevant to the user if the tree node "*covers*" as many microblogs of the user as possible (in other words, it contains candidate locations of many microblogs). Next we formally define the concept of "*coverage*" of node n as follows.

Definition 5 (Coverage): The coverage of node n is the sum of similarities of its covered microblogs, defined as

$$\mathcal{C}(n) = \sum_{1 \le i \le |M|} \mathcal{C}(n, m_i), \tag{3}$$

where $\mathcal{C}(n, m_i)$ is the coverage of node n on microblog m_i, which is the largest similarity of nodes under n, i.e.,

$$\mathcal{C}(n, m_i) = \max_{d \in N(m_i) \cap D(n)} \mathcal{J}(d, m_i), \tag{4}$$

where $D(n)$ denotes the set of n's descendants, $N(m_i)$ is the candidate location set of m_i, and $\mathcal{J}$ is the function $\mathcal{J}_e$ (or $\mathcal{J}_f$) for exact extraction (or fuzzy extraction).

For example, consider microblog m_3. Suppose we use exact matching extraction. m_3 has four candidate locations: 1.1.1.2, 1.1.1.2.2, 1.1.2.2.2, 1.2.3.2. Their similarities to the microblog m_3 are respectively $\frac{1}{5}$, $\frac{2}{5}$, $\frac{1}{6}$, $\frac{1}{5}$. Consider node 1.1.1 ("`Hollywood`"). Node 1.1.1 covers two candidate locations 1.1.1.2 and 1.1.1.2.2. As node 1.1.1.2.2 has the largest similarity, $\mathcal{C}(1.1.1, m_3) = \frac{2}{5}$. Similarly $\mathcal{C}(1.1.1, m_1) = \frac{1}{5}$, $\mathcal{C}(1.1.1, m_2) = \frac{1}{5}$, $\mathcal{C}(1.1.1, m_4) = \frac{2}{5}$, $\mathcal{C}(1.1.1, m_5) = \frac{2}{5}$. It will not cover other microblogs. Thus $\mathcal{C}(1.1.1) = \frac{8}{5}$. Similarly $\mathcal{C}(1.1.2) = \frac{5}{5}$ and $\mathcal{C}(1.2) = \frac{9}{5}$.

Obviously, the larger the coverage of a node is, the node is more relevant to the user location. For example, nodes 1.1.1 and 1.2 are better than node 1.1.2. If we want to select top-k locations from the same level (for example, in some applications we want to identify top-k cities of the user), we can identify k tree nodes with the largest coverage as the top-k locations of the user. However notice that some microblogs covered by a node are also covered other nodes. For example, many microblogs covered by node 1.1.2 are also covered by node 1.1.1. To address this issue, we want to select k nodes that cover as many "*distinct*" microblogs as possible to maximize the overall coverage. Next we formulate the problem.

Definition 6: The problem of identifying top-k locations from the same level is to find the k-node set N_{best} which has the largest overall coverage. That is

$$N_{best} = \operatorname*{argmax}_{N_k} \sum_{1 \le i \le |M|} \max_{n \in N_k} \mathcal{C}(n, m_i), \tag{5}$$

where N_k is any set of k nodes selected from the given level.

We can prove the problem of identifying top-k tree nodes from the same level is NP-hard as formalized in Lemma 1, by a reduction from the weighted set cover problem.

Lemma 1: The top-k user location identification problem from the same level is NP-hard.

Since the problem of identifying top-k locations from the same level is NP-hard, we propose a greedy algorithm. We iteratively select the tree nodes as follows. First, we select node n with the maximum coverage. Next we select the next node with the maximum coverage in the remainder microblogs (which are not covered by the first node). Iteratively, we can select top-k tree nodes. In the algorithm, a challenge is to efficiently compute coverage $\mathcal{C}(n)$. Next we propose an efficient algorithm to address this issue.

Algorithms for computing coverage $\mathcal{C}(n)$. For each microblog m_i, we extract a set of entities. For each entity, we get a sorted list of tree nodes (sorted by Dewey codes). We merge these lists using a merge-join algorithm and generate a sorted list of candidate locations, i.e., $N(m_i)$.

Given any tree node n, its descendants (nodes in $D(n)$) must be in a range $[n, n')$ where $n' = n + 1$ is derived by increasing the last number of n's Dewey code by 1. Since the candidate locations in $N(m_i)$ are also sorted by Dewey codes, we can compute a range $[x, y]$ such that $N(m_i)[x, y] = N(m_i) \cap D(n)$. Notice that we can efficiently compute the range using a binary search algorithm. We first use n's Dewey code to do a binary search on $N(m_i)$ and get the first candidate location that is not smaller than n. Suppose we get $N(m_i)[x]$. Next we use the Dewey code of n' to do a binary search on $N(m_i)$ and get the last candidate node that is smaller than n'. Suppose we get $N(m_i)[y]$. By enumerating candidate locations in $N(m_i)[x, y]$, we can get the maximum similarity between node n and microblog m_i, i.e., $\mathcal{C}(n, m_i)$. By enumerating each microblog, we can compute $\mathcal{C}(n) = \sum \mathcal{C}(n, m_i)$. When the numbers of candidate locations in $N(m_i)[x, y]$ is large, this algorithm is expensive. As such, we propose a range maximum query (RMQ) algorithm to compute $\mathcal{C}(n, m_i)$ efficiently.

For each $u \in [1, |N(m_i)|]$, let $h = \lceil \log(|N(m_i)| - u) \rceil$. We maintain h values, $\mathcal{F}[u, 1], \ldots, \mathcal{F}[u, h]$, where $\mathcal{F}[u, v]$ is the maximal similarity among the locations in $N(m_i)[u, u+2^v]$ to m_i for $1 \le v \le h$. To get the maximal similarity of location candidates in $N(m_i)[x, y]$, we can split $[x, y]$ into two ranges $[x, x + 2^d]$ and $[y - 2^d, y]$, where $d = \lfloor \log_2(y - x) \rfloor$. Since the maximum similarity in these two ranges, i.e., $\mathcal{F}[x, d]$ and $\mathcal{F}[y - 2^d, y]$ are materialized, we can efficiently compute

$$\mathcal{C}(n, m_i) = \max(\mathcal{F}[x, d], \mathcal{F}[y - 2^d, d]).$$

The space complexity of $\mathcal{F}[u, v]$ is $\mathcal{O}(|N(m_i)| \log |N(m_i)|)$. The time complexity to compute $\mathcal{F}[u, v]$ using a dynamic-programming algorithm is $\mathcal{O}(|N(m_i)| \log |N(m_i)|)$. Notice that we materialize the structure once and use it to compute the coverage for many tree nodes. For each node n, the time complexity to locate a range is $\mathcal{O}(\log |N(m_i)|)$ and time complexity to compute $\mathcal{C}(n, m_i)$ is $\mathcal{O}(1)$. Thus the time complexity to compute $\mathcal{C}(n)$ is $\mathcal{O}(\sum_{1 \le i \le |M|} \log |N(m_i)|)$.

For example, consider microblog m_3. Suppose we use the exact matching based method. It has four candidate locations. We get its sorted list $N(m_3) = \{1.1.1.2, 1.1.1.2.2, 1.1.2.2.2, 1.2.3.2\}$. $\mathcal{S}_{m_3} = [\frac{1}{5}, \frac{2}{5}, \frac{1}{6}, \frac{1}{5}]$. Consider node 1.1.1. Its descendants must be in [1.1.1, 1.1.2). To compute $N(m_3) \cap$

$D(1.1.1)$, we use node 1.1.1 to do a binary search in $N(m_3)$ and get $x = 1$. Then we use node 1.1.2 to do a binary search and get $y = 2$. Thus $N(m_3)[1,2] = N(m_3) \cap D(1.1.1)$. To facilitate computing the maximum similarity in a range, we maintain $\mathcal{F}[1,1] = \frac{2}{5}$, $\mathcal{F}[1,2] = \frac{2}{5}$, $\mathcal{F}[2,1] = \frac{2}{5}$, and $\mathcal{F}[3,1] = \frac{1}{5}$. To compute $N(m_3)[1,2]$, we can directly get it from the maintained list $\mathcal{F}[1,1]$ in $\mathcal{O}(1)$ time.

B. Identifying Top-k Locations from Different Levels

The coverage based method in Section VI-A is effective to select top-k locations from the same level. However it cannot support identifying nodes from different levels. This is because the root always has the largest coverage based on Equation 3 since it covers all tree nodes. In other words, even if a tree node can cover many microblogs, it may not be a good location if its region is rather large. In addition, we need to identify top-k locations from different levels in many applications, e.g., city and district. To address this issue, besides considering the *coverage* of a node, we also take into account the *divergence* of microblogs since different microblogs may refer to different locations. To this end, we utilize the information entropy to quantify the divergence of a node, which is defined as below.

Definition 7 (Entropy): Given a node n with a set of children $\text{CHILD}(n) = \{c_1, c_2, \ldots, c_{|n|}\}$, the entropy of node n is defined as

$$\mathcal{H}(n) = -\sum_{i=1}^{i=|n|} P_{c_i} \cdot \ln P_{c_i} \tag{6}$$

where P_{c_i} denotes the probability to select child c_i as a top-k location and $|n|$ denotes the number of children of node n.

There are two challenges in using information entropy to identify locations. The first one is how to compute the probability P_{c_i}. The second challenge is how to efficiently select top-k locations based on the entropy of a node. Next we discuss how to address these two challenges.

Computing the probability P_{c_i} of each child c_i of node n. Intuitively, among these child nodes, the larger coverage of a child, the higher probability the child node will be selected as a candidate location. Thus the probability of selecting a node as a location is in proportion to the coverage of the node. In other words, the probability distribution curve should be consistent with the coverage distribution curve. Thus we can use the coverage to estimate the probability as follows.

Definition 8 (Probability): Given a node n, the probability of selecting its child c_i as a location is

$$P_{c_i} = \frac{\mathcal{C}(c_i)}{\sum_{c_j \in \text{CHILD}(n)} \mathcal{C}(c_j)}. \tag{7}$$

For example, consider node 1.1. It has two children 1.1.1 and 1.1.2. $\mathcal{C}(1.1.1) = \frac{8}{5}$ and $\mathcal{C}(1.1.2) = \frac{5}{5}$. Thus $P_{1.1.1} = \frac{8}{13}$ and $P_{1.1.2} = \frac{5}{13}$. $\mathcal{H}(1.1) = -\frac{8}{13}\log\frac{8}{13} - \frac{5}{13}\log\frac{5}{13} = 0.799$.

Using entropy to select top-k locations. As we know, the larger the information entropy is, the more chaotic the system will be. Thus when the information entropy of node n is too large, the child nodes of node n have similar probabilities (or coverage). That is the microblogs are distributed uniformly across all the children. In this case, we cannot distinguish these children and select n as a top-k location. On the contrary, if the entropy is very small, some children have much larger probabilities (or coverage) than others. That is the microblogs are frequently distributed in some children. In this case, we select some children as top-k locations.

Next we formalize how to decide selecting node n or n's children as top-k locations. When each child node has the same probability, the information entropy of node n reaches the maximum value, denoted by H_{max}, which is calculated as:

$$H_{max} = -\sum_{i=1}^{i=|n|} \frac{1}{|n|} \cdot ln\frac{1}{|n|} = |n| \cdot \frac{1}{|n|} \cdot ln|n| = ln|n|. \tag{8}$$

Given an entropy bound $\mathcal{B} = \varepsilon \cdot H_{max}$, if the entropy of node n is larger than the bound $\mathcal{B}$, we select node n as a top-k location. On the contrary if $\mathcal{H}(n) \leq \mathcal{B}$, we select top-k locations from n's children by checking its children's entropy.

Next we formulate the problem of using the entropy and coverage to identify top-k locations from different levels.

Definition 9: The problem of identifying top-k locations from different levels is to find the k-node set N_{best} to maximize the coverage subject to the entropy constraint. That is

$$N_{best} = \underset{N_k}{\text{argmax}} \sum_{1 \leq i \leq |M|} \max_{n \in N_k} \mathcal{C}(n, m_i), \tag{9}$$

subject to for each node $n \in N_k$, $\mathcal{H}(n) > \mathcal{B}$.

The problem of identifying top-k locations from different levels is also NP-hard as formalized in Lemma 2.

Lemma 2: The top-k user location identification problem from different levels is NP-hard.

Algorithms for computing top-k locations: Since the problem is NP-hard, we propose a greedy algorithm. A straightforward method enumerates every node, removes the nodes that invalidate the entropy condition, and identifies the k nodes with the largest coverage in the remainder nodes. However this method needs to enumerate every node and thus is expensive. To improve the performance, we propose a best-first algorithm. We first check the root, compute its coverage and entropy, and add it into a priority queue. Next we always pop the node with the maximum coverage from the queue. If the entropy of the node is larger than the bound $\mathcal{B}$, we select it as a top-k location and add it into the result set $\mathcal{R}_u$. If there are k locations in $\mathcal{R}_u$, we terminate the algorithm. On the contrary, if the entropy of the node is not larger than the bound $\mathcal{B}$, we access its children, compute their coverage and insert them into the priority queue. Iteratively, we can identify the top-k locations. Figure 4 shows the pseudo-code of our algorithm.

For example, consider the microblogs in Table I and the location-based tree structure in Figure 1. We first add the root into the priority queue $\mathcal{Q}$. Then we pop it from $\mathcal{Q}$. Since its entropy is not larger than the bound $\mathcal{B} = 0.8$, we access its children 1.1 and 1.2. As node 1.2 has the largest coverage, we pop node 1.2. We compute its entropy which is larger than the bound $\mathcal{B}$. Since it is a top-k location, we add it into the result set. Next we pop node 1.1. As its entropy is not larger than the bound $\mathcal{B}$, we add its two child nodes 1.1.1 and 1.1.2 into $\mathcal{Q}$. Node 1.1.1 has the largest coverage, we pop node 1.1.1. Since

Algorithm 3: GLITTER-Aggregation ($\mathcal{L}$, k, T)

Input: $\mathcal{L} = \{N(m_i)\}$: Candidate nodes of each microblog; k: An integer; T : Location structure
Output: $\mathcal{R}_u$: The top-k user locations;

```
1  begin
2      Initialize priority queue Q=φ and result set R_u=φ;
3      Q.Enqueue(⟨root, C(root)⟩) ;
4      while R_u.Size() < k & Q is not empty do
5          n ← Q.Dequeue() ;
6          if H(n) > B then Insert n into R_u;
7          else
8              foreach child node c ∈ CHILD(n) do
9                  Q.Enqueue(⟨c, C(c)⟩) ;
10 end
```

Fig. 4. GLITTER-Aggregation Algorithm(Identifying top-k User Locations).

its entropy is larger than the bound $\mathcal{B}$, it is a top-k location. If we want to find top-2 locations, nodes 1.2 and 1.1.1 are the results, and we terminate the algorithm.

VII. LOCATION REFINEMENT

In this section, we study how to refine the candidate locations and identify top-k locations of each microblog. We first propose a refinement model to refine the locations of a microblog of a user based on other microblogs the user has posted in Section VII-A and then devise an efficient algorithm to identify top-k locations of a microblog in Section VII-B.

A. Refinement Model

Basic Idea. Let $\mathcal{R}_u$ denote the top-k locations of a user, which is identified in the aggregation step. Obviously locations in $\mathcal{R}_u$ are relevant to multiple microblogs. However these locations are derived from aggregating location information from multiple microblogs and may not appear in any single microblog. We can add them into the corresponding microblogs to facilitate identifying top-k locations of microblogs more accurately.

We use an example to show our basic idea. Consider the microblogs in Table I. The top-2 user locations are "`Hollywood`" (1.1.1) and "`San Diego`" (1.2). Consider microblog m_8 = "Bought shoes from `Sports Shop Center`". The candidate locations of m_8 are nodes 1.1.2.1.1 and 1.2.3.1.1. These two nodes have the same similarity with m_8. However most of the other microblogs of the user are covered by "`San Diego`" (1.2). Thus we can deduce that the user may omit "`San Diego`" in microblog m_8. To better rank the candidate locations, we want to add "`San Diego`" into m_8 and recompute the similarity between m_8 and node 1.2.3.1.1. Using this method, node 1.2.3.1.1 will have larger similarity with m_8 than node 1.1.2.1.1, and should be the best location of m_8. Next we formally introduce our refinement model.

Refinement Model. For each microblog m_i posted by the user, we enumerate each candidate location in $N(m_i)$. For each candidate location, we check whether there exists a location in $\mathcal{R}_u$ which is an ancestor of the candidate location. If yes, we add the entity of the location into $E(m_i)$ for exact matching; or we add the pair $\langle e, e\rangle$ into $E'(m_i)$ for fuzzy matching where e is the entity of the location. Then we recompute the similarity for the candidate location and the microblog as follows. (1) For exact matching, we utilize Equation 1 with the updated $E(m_i)$ to compute the similarity between m_i and each candidate location. (2) For fuzzy matching, we utilize Equation 2 with the updated $E'(m_i)$ to compute the similarity between m_i and each candidate location. Based on the refined similarity of every candidate node, we select the k best candidate nodes with the largest similarity as the top-k locations of the microblog.

For example, consider microblog m_8. Its candidate nodes are 1.1.2.1.1 and 1.2.3.1.1. $E(m_8)$ = "`Sports Shop Center`". The top-2 user locations are "`Hollywood`" (1.1.1) and "`San Diego`" (1.2). For node 1.2.3.1.1, node 1.2 is its ancestor, and thus we add its corresponding entity into $E(m_8)$. Thus the updated set $E^u(m_8)$= {`Sports Shop Center`, `San Diego`}. Then we compute the similarity between m_8 and node 1.2.3.1.1 based on Equation 1. We have $\mathcal{J}_e(m_8, 1.2.3.1.1) = \frac{2}{5}$. For node 1.1.2.1.1, both of the top-2 user locations are not its ancestor. Thus $\mathcal{J}_e(m_8, 1.1.2.1.1)$ is still $\frac{1}{5}$. If we want to identify top-1 location of m_8, node 1.2.3.1.1 is the best answer.

To identify the top-k locations of a microblog, a naive method enumerates each candidate location and each user location. However when the number of candidate locations is large, this method is inefficient. In the following, we propose an efficient refinement algorithm.

B. Efficient Refinement Algorithm

Given a microblog m_i, we can prune many candidate locations that have small similarities to m_i. Based on this idea, we devise a pruning rule (Lemma 3): For each candidate location n, its refined similarity S' will be larger or equal to the original similarity S. If there is no user location which is an ancestor of n, $S' = S$. If there exists a user location which is an ancestor of n, $S' - S \leq \frac{1}{|E(m_i) \cup E(n)|}$ for exact matching and $S' - S \leq \frac{1}{|\pi(E'(m_i)) \cup E(n)|}$ for fuzzy matching.

Lemma 3: Given a microblog m_i and a candidate location n, suppose the original similarity is S and the refined similarity is S'. We have

(1) $S' - S \leq \frac{1}{|E(m_i) \cup E(n)|}$ for exact matching;

(2) $S' - S \leq \frac{1}{|\pi(E'(m_i)) \cup E(n)|}$ for fuzzy matching.

Based on this idea, we devise an efficient refinement algorithm. Figure 5 shows the pseudo-code. Given a microblog m_i, we first sort its candidate locations by their similarity to m_i in descending order and then access the candidate locations in order. We take the similarity of the k-th node as a lower bound τ. For each candidate location n, if its original similarity is smaller than $\tau - \frac{1}{|E(m_i) \cup E(n)|}$ for exact matching or $\tau - \frac{1}{|\pi(E'(m_i)) \cup E(n)|}$ for fuzzy matching, we can terminate; otherwise we check whether there exists a user location which is an ancestor of this candidate location. If yes, we recompute the similarity and update the lower bound. Iteratively, we can compute the top-k locations.

VIII. SUPPORTING UPDATES

A. Microblog Updates

Since microblogs will be frequently updated, we propose an incremental algorithm to efficiently support updates of

Algorithm 4: GLITTER-REFINE ($\mathcal{R}_u$, k, m_i, T)

Input: $\mathcal{R}_u$: The top-k user locations; k: An integer; m_i : A microblog; T : Location structure
Output: $\mathcal{R}_{m_i}$: The top-k locations of m_i;

```
1  begin
2      Sort candidate locations of m_i;
3      S_τ = similarity of the k-th location ;
4      foreach candidate location n in order do
5          S = Similarity between m_i and n;
6          if S ≤ τ − 1/|E(m_i)∪E(n)| then return;
               // Fuzzy: S ≤ τ − 1/|π(E'(m_i))∪E(n)|
7          if A location in R_u is an ancestor of n and the
           entity is not in E(m_i) then
8              S+ = 1/|E(m_i)∪E(n)| ;
               // Fuzzy: S+ = 1/|π(E'(m_i))∪E(n)|
9          if S ≥ S_τ then
10             R_{m_i} ← n and update S_τ, R_{m_i} ;
11 end
```

Fig. 5. GLITTER-REFINE Algorithm of Refining Locations of Microblogs.

microblogs. Consider a user with a set of microblogs M. Suppose the user posts another set of microblogs Δ_M. We want to identify the top-k locations of the user on $M + \Delta_M$ and the top-k locations for each microblog in Δ_M. (Although we can identify time-dependent locations, we leave it as a future work.) A naive method is to apply our algorithms on $M + \Delta_M$. However this method cannot utilize the identified results on M. As such, we propose an incremental algorithm.

When identifying locations on M of the user, we maintain a location subtree for the user. The root of the substree is the same as the tree-based location structure. The root has k children which are the top-k countries computed from M. For each such country, it has k children which are the top-k states computed from M. Similarly, we add the top-k city nodes and top-k district nodes into the subtree. For each node n, we keep its coverage $\mathcal{C}(n)$ and entropy $\mathcal{H}(n)$. As the subtree has at most 5 levels, it is very small (at most 4000 nodes).

Then, for each microblog in Δ_M, we first extract location entities from the subtree. If we find entities from the subtree, we will utilize them to find candidate locations; otherwise we use the tree-based location structure to identify location entities using our proposed method. To support extraction on the substree, we need to add the following index structures. For exact matching, we maintain a hash table for each of top-k countries of the user. Then we extract the locations from the top-k countries for each microblog. Based on the locality principle, microblogs of a user will be located together in some regions. Thus we can utilize this method to improve the performance. For fuzzy matching, we use a similar way and maintain segment-based indexes. Notice that fuzzy extraction is expensive on large datasets. Thus this locality based method can significantly improve the performance.

Based on the extracted candidate locations, we update coverage $\mathcal{C}(n)$ and entropy $\mathcal{H}(n)$ of each node. Then we recompute top-k locations using the aggregation algorithm. Based on top-k user locations, we use the refinement algorithm to compute the top-k locations of each microblog in Δ_M.

B. Tree-based Location Structure Updates

When the location hierarchy and POIs are updated, we need to update the tree-based location structure, and the inverted index for exact matching (and segment-based inverted index for fuzzy matching). If a POI is newly inserted, we first insert the POI into the tree-based location structure in an up-down manner from the root. If the update introduces new tree nodes, we need to assign new Dewey codes to the inserted nodes. Notice that there are many update-aware Dewey coding techniques [24] and we do not need to rebuild the Dewey codes. If there are new entities, we need to insert them into the inverted index (and segment-based index for fuzzy matching). Similarly, if the update needs to remove some tree nodes, we can use similar techniques to implement the updates. For the updates of location hierarchy and POIs, we can use a first-delete-then-insert strategy.

IX. EXPERIMENTS

In this section, we report results of an experimental study to evaluate our proposed method. The goal of our study is to evaluate (1) the quality of top-k user locations and top-k microblog locations identified based on our algorithms; and (2) the running time to identify locations.

Datasets: We used two real-world datasets: `Twitter` and `Foursquare`. For `Twitter`, we used 3.5 thousand users and 2.25 million microblogs. Each user had about 600 microblogs. For `Foursquare`, we used 21,000 users and 1 million check-ins (For simplicity, check-ins are also called microblogs). Each user had 48 microblogs. Table II shows the details.

TABLE II. DATASETS.

	# Users	# Microblogs	# Microblogs/User
`Twitter`	3547	2,254,338	635
`Foursquare`	21,344	1,028,672	48

Experimental Setting: All the algorithms were implemented in Java. All the experiments were conducted in a machine with 2.5GHz Inter 8-core CPU, 16GB RAM, running Ubuntu operating system 10.04.

A. Evaluating Quality

In this section, we evaluate the location quality. We first consider exact matching on the `Twitter` dataset and will show the results for fuzzy matching and on the `Foursquare` dataset in Section IX-C. We use the well-known metrics: precision, recall and F-measure to evaluate the quality, which will be defined in the following sections.

1) Evaluating Quality of Microblog Locations: We first evaluate the quality of top-k locations of a microblog.

Evaluation Metrics: To evaluate the microblog quality, we selected microblogs with latitude and longitude which can be used to identify the real locations. For each microblog, if the distance between the real location and one of the top-k locations computed by our algorithm was smaller than a threshold (100 meters), our method was regarded as correctly finding the real location. In this way we can compute precision and recall. The recall is the ratio of the number of microblogs with correctly identified locations to the total number of microblogs. The precision is the ratio of the number of microblogs with correctly identified locations to the number of microblogs with

identified locations. F-measure is the weighted harmonic mean of precision and recall.

We implemented two algorithms: (1) Extraction [6], traditional extraction based method which extracted the candidate locations and selected the k best locations with the largest similarities as top-k locations. (2) Extraction+Refinement, which used the refinement model to refine the candidate locations and identified top-k locations. Since the results for different k values are similar, we only report the results for $k = 3$ due to space constraints. Figure 6 shows the results.

We can see that Extraction+Refinement significantly outperformed Extraction, especially on recall. This is because many microblogs included "incomplete" locations and existing extraction based methods cannot identify their real locations based on "local" microblogs. Extraction+Refinement aggregated multiple microblogs and can refine the local microblogs based on global results. For example, the recall of Extraction was smaller than 0.6 and that of Extraction+Refinement nearly reached 0.9. This is contributed by our global identification model and the refinement model which considered multiple microblogs.

2) Evaluating Quality of User Locations: We then evaluate the quality of top-k locations of a user.

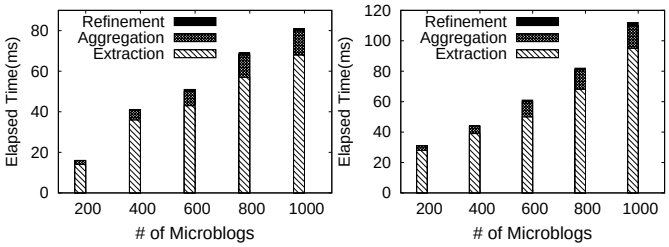
Evaluation Metrics: Evaluating a user's location is subjective and only the user can give her interested locations. As there is no standard benchmark and it is non-trivial to identify user's real locations, we utilized user study. We randomly selected 1000 users (strangers to us) across the world whose microblogs are related to more than 5 different cities. We asked them to give their top-k interested locations and take the returned results as the ground truth. However only about 100 of them returned results. Based on these returned locations, we compute the precision and recall. The precision is the ratio of the overlap size of the real locations and the identified locations by the algorithms to the number of identified locations by the algorithms. The recall is the ratio of the overlap size of the real locations and the identified locations by the algorithms to the number of real locations by user study. (The number of identified locations and the number of real locations may be different as an algorithm may return fewer than k locations.)

We implemented three algorithms: (1) SameLevel-City, which identifies user locations in the city level (e.g., `Los Angeles`); (2) SameLevel-District, which identifies user locations in the district level (e.g., `Manhattan`). (3) DiffLevel, which identifies top-k locations in different levels. Figure 7 shows the results. We can see that DiffLevel outperformed SameLevel-City and SameLevel-District significantly, especially on recall. The recall of DiffLevel reached 0.9 while that of SameLevel-City was 0.3. This is because the top-k locations of a user may be from different levels. In some cases a user location may be a city (e.g., `Los Angeles`) and in other cases a user location should be a district (e.g., `Manhattan`).

We also conduct an experiment to validate a user's top-1 location for the users providing with city-level locations. Our method still achieves more than 85% precision and recall. Due to space constraints, we do not show the details.

B. Evaluating Efficiency

In this section, we evaluate the efficiency. Our method includes three steps: Extraction, Aggregation, Refinement. We evaluated the average running time for each user in three steps. We tested both exact matching and fuzzy matching. In the paper, for fuzzy matching, we used the normalized edit similarity to quantify the similarity between locations and microblogs and the threshold was 0.9. Figure 8 shows the average running time for microblogs of each user.



(a) Exact Match. (b) Fuzzy Match.

Fig. 8. Evaluating performance on `Twitter` dataset ($k = 3$).

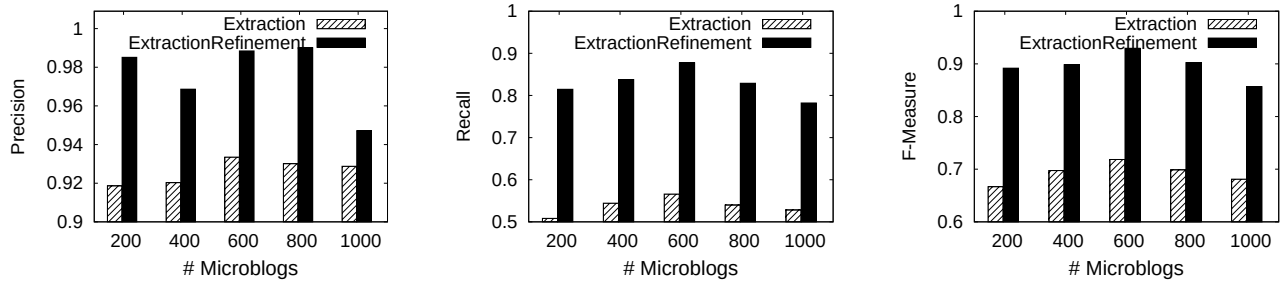
We can see that our method is very efficient. For exact matching, the total running time was about 80 milliseconds for 1000 microblogs of a user. For fuzzy matching, the time was about 120 milliseconds. Another observation is that the extraction time was the dominant time (about 80-90% of the total time). In other words, the aggregation and refinement steps took little time, because our aggregation and refinement algorithms were very efficient.

C. Comparing with Existing Methods

In this section, we compare our algorithms with state-of-the-art methods, content-based method (`Content` [9]) and the unified discriminative influence model (`UDI` [17]), in terms of both quality and performance. The former uses a probabilistic model to identify city-level locations based on content of microblogs and the latter utilizes the relationships between users to identify locations. Notice that these methods only focused on identifying user locations and cannot identify locations of a microblog.

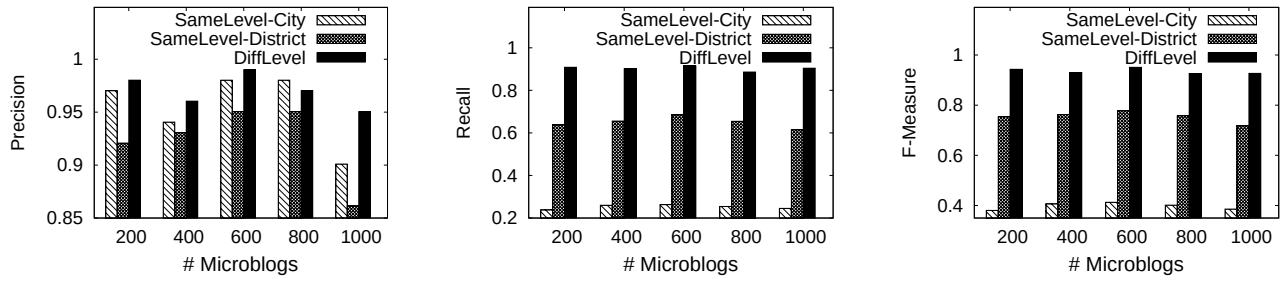
1) Comparing Quality: In this section, we compare the quality and Figures 9 and 10 respectively show the results on the `Twitter` and `Foursquare` datasets. We can see that fuzzy matching indeed improved the quality since it tolerated inconsistencies between microblogs and location entities. Thus it can improve the recall. Another observation is that our method significantly outperformed state-of-the-art methods. The main reason is that (1) they identified the city-level locations and cannot identify districts or business regions; and (2) they cannot combine location entities from different microblogs to identify user locations. For example, on the `Twitter` dataset, the F-measure of `Content` and `UDI` were respectively 0.55 and 0.75 while our exact-matching method improved it to 0.9 and our fuzzy-matching method further improved it to 0.95. In addition, `UDI` achieved higher recall than `Content` since `UDI` utilized relationships between users to identify locations. In addition, state-of-the-art methods achieved better quality on `Foursquare` than `Twitter`, because `Foursquare` contained higher quality location entities. Our method achieved high quality on both of the two datasets as we aggregated multiple microblogs to identify locations.

2) Comparing Performance: In this section, we compared the performance of our methods with state-of-the-art methods. Figure 11 shows the average running time for each user.



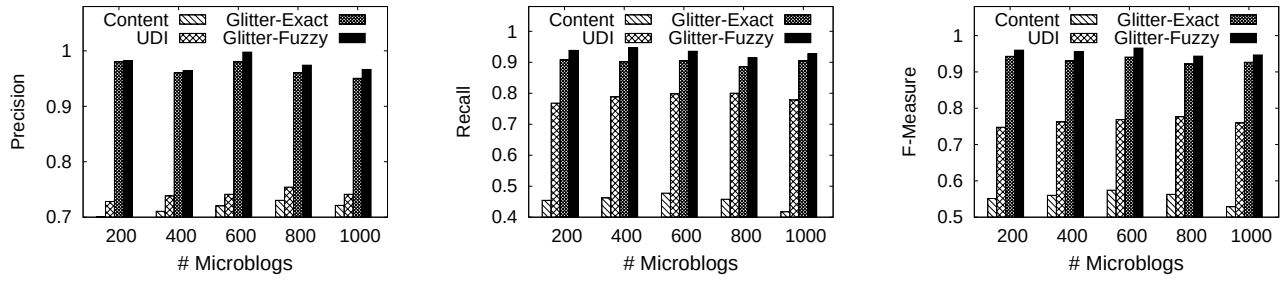
(a) Precision. (b) Recall. (c) F-measure.

Fig. 6. Evaluating quality of top-k locations of microblogs on `Twitter` dataset (exact match, $k = 3$).



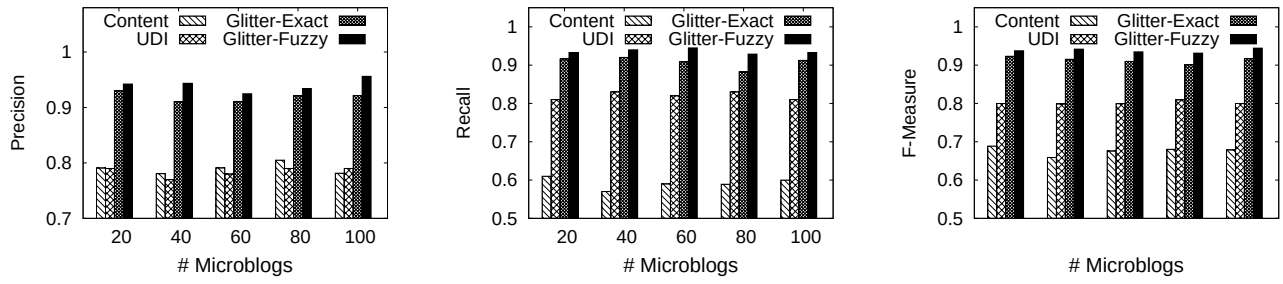
(a) Precision. (b) Recall. (c) F-measure.

Fig. 7. Evaluating quality of top-k locations of users on `Twitter` dataset (exact match, $k = 3$).



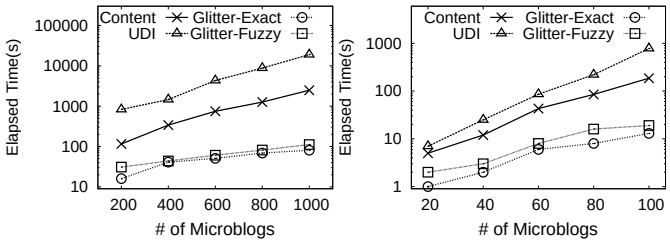
(a) Precision. (b) Recall. (c) F-measure.

Fig. 9. Comparing quality of top-k locations of users on `Twitter` dataset ($k = 3$).



(a) Precision. (b) Recall. (c) F-measure.

Fig. 10. Comparing quality of top-k locations of users on `Foursquare` dataset ($k = 3$).



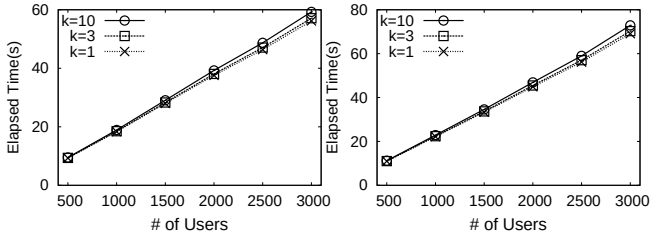
(a) `Twitter`. (b) `Foursquare`.

Fig. 11. Comparing performance of top-k locations of users ($k = 3$).

Our method significantly outperformed existing methods, even by 2 orders of magnitude. For example, on the `Twitter` dataset, for 1000 microblogs, existing method took more than 10,000 milliseconds, while our two methods only took about 100 milliseconds. The main reason is that they used machine-learning based techniques which are rather expensive while we devised efficient entity extraction, aggregation and refinement algorithms which are very efficient. In addition, `UDI` was slower than `Content` as `UDI` used much more information (e.g., relationships between different users) than `Content`. Notice that all of these algorithms achieved higher performance on the `Foursquare` dataset, because in the `Foursquare` dataset, each user contained smaller numbers of microblogs.

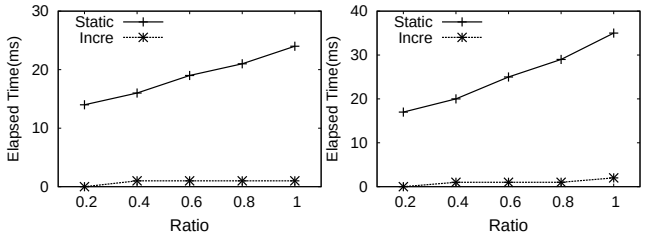
D. Scalability

In this section, we evaluate the scalability of our method on the `Twitter` dataset. We varied the number of users and evaluated the total running time of identifying the locations for these users. We also tested different k values. Figure 12 shows the results. We can see that our method scaled very well with the increase of the number of users. Our method achieved similar results on different k values. For example, on



(a) Exact Match. (b) Fuzzy Match.

Fig. 12. Scalability on `Twitter` dataset.



(a) Exact Match. (b) Fuzzy Match.

Fig. 13. Updates on `Twitter` dataset ($k = 3$).

fuzzy matching, for 500 users, our method took 12 seconds and the time increased to 70 seconds for 3000 users.

E. Updates

In this section, we evaluate the effects of updates of microblogs. We implemented two algorithms: (1) Static, which computed the results from scratch; (2) Incre, which computed the results incrementally. Given a user, we first used her 50% microblogs as the original dataset, and then we added another 10% more microblogs into the dataset and evaluated the running time. We repeated these steps in five times. Figure 13 shows the results. We can see that Incre was superior over Static. There are two reasons. First, Incre did not need to identify locations from scratch. Second, Incre can utilize the results of the original datasets to prune unnecessary locations which can help improve the extraction performance.

We conducted experiments to evaluate updates of location structure. For insertions of 1000 new POIs, it took about 30 milliseconds. Due to space constraints, we omit the results.

X. Conclusion

We have studied the problem of identifying top-k locations of a microblog and top-k locations of a user. We proposed a three step method. The extraction step extracts location entities from the microblogs. The aggregation step aggregates the candidate locations and generates top-k locations of the user. The refinement step refines the candidate locations using the top-k user locations and computes the top-k locations of each microblog. We developed incremental algorithm to efficiently support dynamic updates of microblogs. Experiments on real-world datasets showed that our method achieves high quality and good performance, and scales very well.

Acknowledgement. This work was partly supported by the National Natural Science Foundation of China under Grant No. 61272090 and 61373024, National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, Beijing Higher Education Young Elite Teacher Project under grant No. YETP0105, a project of Tsinghua University under Grant No. 20111081073, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the "NExT Research Center" funded by MDA, Singapore, under Grant No. WBS:R-252-300-001-490.

References

[1] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable ad-hoc entity extraction from text collections. *PVLDB*, 1(1):945–957, 2008.

[2] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *SIGIR*, pages 273–280, 2004.

[3] L. Backstrom, J. M. Kleinberg, R. Kumar, and J. Novak. Spatial variation in search engine queries. In *WWW*, pages 357–366, 2008.

[4] L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. In *WWW*, pages 61–70, 2010.

[5] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin. An efficient filter for approximate membership checking. In *SIGMOD Conference*, pages 805–818, 2008.

[6] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient batch top-k search for dictionary-based entity recognition. In *ICDE*, pages 28–39, 2006.

[7] S. Chandra, L. Khan, and F. B. Muhaya. Estimating twitter user location using social interactions-a content based approach. In *SocialCom/PASSAT*, pages 838–843, 2011.

[8] S. Chaudhuri, V. Ganti, and D. Xin. Mining document collections to facilitate accurate approximate entity matching. *PVLDB*, 2(1):395–406, 2009.

[9] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *CIKM*, pages 759–768, 2010.

[10] D. Deng, G. Li, and J. Feng. An efficient trie-based method for approximate entity extraction with edit-distance constraints. In *ICDE*, pages 141–152, 2012.

[11] Q. Hao, R. Cai, C. Wang, R. Xiao, J.-M. Yang, Y. Pang, and L. Zhang. Equip tourists with knowledge mined from travelogues. In *WWW*, pages 401–410, 2010.

[12] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[13] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *WWW*, pages 769–778, 2012.

[14] G. Li, D. Deng, and J. Feng. Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction. In *SIGMOD Conference*, pages 529–540, 2011.

[15] G. Li, D. Deng, and J. Feng. A partition-based method for string similarity joins with edit-distance constraints. In *ACM TODS*, 2013.

[16] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: A partition-based method for similarity joins. *PVLDB*, 5(3):253–264, 2011.

[17] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *KDD*, pages 1023–1031, 2012.

[18] J. Lu, J. Han, and X. Meng. Efficient algorithms for approximate member extraction using signature-based inverted lists. In *CIKM*, pages 315–324, 2009.

[19] Q. Mei, C. Liu, H. Su, and C. Zhai. A probabilistic approach to spatiotemporal theme pattern mining on weblogs. In *WWW*, pages 533–542, 2006.

[20] T. Rattenbury, N. Good, and M. Naaman. Towards automatic extraction of event and place semantics from flickr tags. In *SIGIR*, pages 103–110, 2007.

[21] C. Sun and J. F. Naughton. The token distribution filter for approximate string membership. In *WebDB*, 2011.

[22] J. Wang, G. Li, and J. Feng. Trie-Join: Efficient Trie-based String Similarity Joins with Edit-Distance Constraints constraints. In *PVLDB*, 3(1):1219-1230, 2010.

[23] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *SIGMOD Conference*, pages 759–770, 2009.

[24] L. Xu, T. W. Lin, H Wu, and Z. Bao. DDE: from dewey to a fully dynamic XML labeling scheme In *SIGMOD Conference*, pages 719-730, 2009.

[25] Z. Yin, L. Cao, J. Han, C. Zhai, and T. S. Huang. Geographical topic discovery and comparison. In *WWW*, pages 247–256, 2011.