Microblog Entity Linking with Social Temporal Context

Wen Hua [§†], Kai Zheng [§‡], Xiaofang Zhou [§‡]
[§] School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia
[†] School of Information, Renmin University of China, Beijing, China
[‡] School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu, China
w.hua@uq.edu.au {kevinz, zxf}@itee.uq.edu.au

ABSTRACT

Nowadays microblogging sites, such as Twitter and Chinese Sina Weibo, have established themselves as an invaluable information source, which provides a huge collection of manually-generated tweets with broad range of topics from daily life to breaking news. Entity linking is indispensable for understanding and maintaining such information, which in turn facilitates many real-world applications such as tweet clustering and classification, personalized microblog search, and so forth. However, tweets are short, informal and error-prone, rendering traditional approaches for entity linking in documents largely inapplicable. Recent work addresses this problem by utilising information from other tweets and linking entities in a batch manner. Nevertheless, the high computational complexity makes this approach infeasible for real-time applications given the high arrival rate of tweets. In this paper, we propose an efficient solution to link entities in tweets by analyzing their social and temporal context. Our proposed framework takes into consideration three features, namely entity popularity, entity recency, and user interest information embedded in social interactions to assist the entity linking task. Effective indexing structures along with incremental algorithms have also been developed to reduce the computation and maintenance costs of our approach. Experimental results based on real tweet datasets verify the effectiveness and efficiency of our proposals.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

General Terms

Algorithms; Experimentation

Keywords

Microblog entity linking; Social temporal context; Entity popularity; Entity recency; User interest

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2751522.

1. INTRODUCTION

Twitter[1], an online social networking and microblogging service, has attracted worldwide attention since its birth in March 2006. With more than 500 million registered users in 2012, it received over 340 million tweets per day[2] about topics ranging from daily life to breaking news. This huge collection of tweets make Twitter an invaluable information source, as well as a platform to maintain social interactions. Obviously, a better understanding of the content published on Twitter can benifit many real-world applications such as tweet clustering and classification, personalized microblog search, and so forth.

Consider microblog search as a motivation example throughout the paper. One of the most important issues faced by microblog search is **entity ambiguity**. Teevan et al. [1], based on analysis on large-scale query logs, observed that users search Twitter mostly to find information about breaking news or people of interest (e.g., celebrities), both of which involve named entities. Meanwhile, according to the statistics of a randomly sampled and manually annotated tweet corpus used in [2], about 45.08% of tweets have at least one named entity mention. In short, queries and tweets are full of named entities that, however, are usually ambiguous in practice, i.e., the same entity mention can correspond to multiple real-world entities. Therefore identifying the exact references for named entities can bring tremendous benefit to improving the accuracy and user experience of microblog search. Consider the example shown in Fig. 1, wherein the leftmost part are entity mentions occurring in tweets (e.g., "Jordan"), in the middle are named entities in a knowledgebase (e.g., *Jordan (country)*, *Air Jordan*, *Michael Jordan (basketball)*, and *Michael Jordan (machine learning expert)*), and each user u_i posts multiple tweets d_j at time t_j. Given a query with some entity mentions (e.g., "Jordan"), if we can successfully link them to the right entities (e.g., *Michael Jordan (basketball)*), the corresponding tweets (e.g., d_5, d_{13} and d_{15}) can be retrieved as the personalized search result. As we will show shortly, the performance of existing entity linking methodologies still needs to be improved, which is the focus of this work.

Next we will first review the state-of-the-art development in entity linking problem with a brief summary of differences between our proposal and existing approaches, and then bring out the challenges and our contributions.

1.1 Literature of Entity Linking

Entity linking is a task of automatically linking entity mentions to real-world entities stored in large-scale machine-understandable knowledgebases (e.g., Wikipedia[3]). Recently, significant progresses

[1] https://twitter.com/
[2] http://en.wikipedia.org/wiki/Twitter
[3] http://www.wikipedia.org/

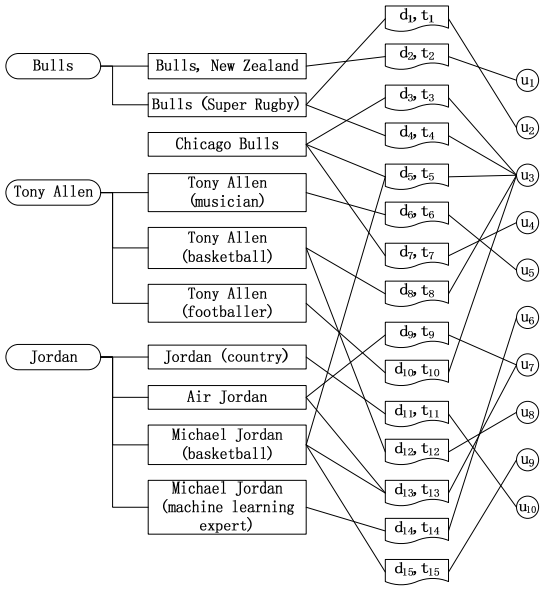


Figure 1: Entity linking for personalized microblog search.

have been achieved in linking named entities detected from Web documents with knowledgebases [3, 4, 5, 6, 7, 8, 9]. Many of them [3, 4] measure mention-entity correspondence by context similarity, i.e., similarity between text around the entity mention and the document describing the entity in a knowledgebase. However, tweets in microblogging sites are usually too short to have sufficient information to calculate context similarity accurately, which leads to unsatisfactory performance. Other approaches [5, 6, 7, 8, 9] assume that entities mentioned in a single document are likely to be topically coherent, and can collectively link mentions in a document by considering the interdependence between corresponding entities. Nevertheless, in the microblogging environment, there are usually too few available mentions to derive a joint and interdependent entity assignment due to the word limitation per tweet.

Recently, increasing attention has been paid to entity linking in short texts like tweets [10, 11, 12, 2, 13]. Regardless of their improvements over methods that link entities in a single tweet by considering topical coherence [14], these approaches are still not flawless. First, most of them [11, 12, 2] utilise information obtained from other tweets, and disambiguate entity mentions in these tweets collectively. They leverage the feature that Twitter users are content generators and hence their interest, which is helpful for entity linking, might be scattered in messages they broadcast. However, the topics of users' tweets vary significantly, making it difficult to detect user interest from such a diverse stream of tweets. Moreover, a large amount of Twitter users are information seekers (rather than content generators) who rarely tweet [15], which also increases the difficulty of learning their interest. Second, some work [13] enriches contextual information of tweets with knowledge from external sources such as Web documents. While such a strategy is feasible in an offline setting, it cannot handle the online tasks such as microblog search where thousands of tweets per second are to be processed in real time. Last but not least, none of them considers the popularity and recency of entities, which are important for capturing the dynamic nature of microblogging environment. In other words, their approaches always link *Michael Jordan* in queries or tweets issued by a user interested in computer science to *Michael Jordan (machine learning expert)*. This is obviously unreasonable, since *Michael Jordan (basketball)* is so popular and well-known that even machine learning experts sometimes talk about him. Furthermore, users' interest can be influenced by recent events and change over time. For example, *Michael Jordan (basketball)* is more likely to be mentioned during NBA seasons while *Michael Jordan (machine learning expert)* is probably a better candidate entity when ICML (International Conference on Machine Learning) is being held.

Our proposed method differs from existing approaches mainly in the following aspects.

- Instead of estimating user interest from her broadcasting history that is widely adopted in existing content-based methods, we resort to social interactions between users, namely the followee-follower relationship. This avoids the difficulty and inaccuracy of modeling user interest from limited amount of tweets with diverse topics.

- We propose a novel feature, entity recency, to model the evolving user interest influenced by recent events in the real world, which is important to capture the dynamic nature of microblogging sites.

- Different from existing collective entity linking methods, our framework links entities independently without considering any intra-tweet or inter-tweet relationships. This makes our approach much more efficient and suitable for online applications with real-time requirement.

1.2 Challenges and Contributions

In this paper, we propose to disambiguate entity mentions derived from microblog queries or tweets by leveraging several contextual information, namely user interest, entity popularity, and entity recency. In order to maintain the accuracy and efficiency of our framework, we need to overcome several challenges as elaborated in below.

Challenge 1. The first challenge is concerned with *measuring user interest by social interactions*. After offline entity linking using state-of-the-art approaches [2], entities in the knowledgebase are linked to tweets along with their timestamps and authorship. In other words, each entity is complemented with a community, namely a collection of users mentioning that entity. For example in Fig. 1, the communities associated with *Michael Jordan (basketball)* and *Michael Jordan (machine learning expert)* are $\{u_3, u_7, u_9\}$ and $\{u_6\}$ respectively. Therefore, measuring a user's interest in a candidate entity is equivalent to measuring her interest in the corresponding community. A straightforward method to estimate a user's interest in a community is to check her reachability with every user in that community through her social network. However, this approach has several drawbacks. First, conventional reachability test only considers connectivity between users. The *small-world* theory indicates that users in Twitter can reach each other within 4.12 hops in average [16], implying that just "reachable" does not mean "interested". Second, it is unreasonable to treat users in a community equally, since different people have different influences to a community, and a user's interest in influential people contributes more to her interest in that community. Third, reachability test between random users is time consuming in large-scale followee-follower network. Fourth, it is very expensive to aggregate reachability between the target user and every other user in a community, given that communities associated with popular entities are extremely large.

Contribution. To tackle the first challenge, we propose a novel concept of weighted reachability to model a user's personal interest

in others. Instead of merely checking whether a path exists between two users, we also consider the number of paths in-between as well as the length of each path. We design indexing structures along with incremental algorithms to reduce the time cost for weighted reachability evaluation over the entire followee-follower network. Moreover, effective ways to estimate user influence in a community and incorporate it into measuring a user's interest in that community are also proposed.

Challenge 2. The second challenge is related to *gathering entity recency from tweets*. Entity recency is modeled by a burst of tweets talking about that entity during recent time period. In other words, entity recency represents an entity's recent popularity, so it is infeasible to pre-calculate and store entity recency offline. Another issue concerning entity recency estimation is that recency could be reinforced between related entities. For example, there might not be any tweets about *Michael Jordan (machine learning expert)* at the beginning of *ICML*. However, the possibility that users search for or tweet about *Michael Jordan (machine learning expert)* in the near future cannot be ignored. It is quite challenging to develop an efficient algorithm to evaluate entity recency on-the-fly while taking their mutual reinforcement into consideration. We capture this phenomenon with recency propagation, through which any change in some entity's recency can affect many other related ones.

Contribution. To cope with above challenge, we measure topical relatedness between entities based on their hyperlinks in the knowledgebase, with the rational that highly related entities should propagate recency to each other in a larger extent. To overcome the efficiency issue, we detect clusters of entities with strong connection, and ignore recency propagation between less relevant ones to avoid expensive recency diffusion. A PageRank-style algorithm has been proposed to conduct recency propagation, which can incorporate both recency gathered from underlying tweets and that reinforced by other highly related entities.

The rest of this paper is organized as follows: in Sec. 2, we briefly summarize development in the literature of graph reachability which is quite related with our work; then we define the problem of entity linking formally in Sec. 3, along with a brief introduction of notations adopted in this work; our approaches and experiments are described in Sec. 4 and Sec. 5 respectively, followed by a conclusion and discussion of future work in Sec. 6. The appendix includes a more detailed review of existing work on named entity recognition, two theorem proofs, dataset descriptions and two additional sets of experiments. At the end we also put a discussion about the issues of handling new entities/meanings and our thoughts of preliminary solution.

2. RELATED WORK

Reachability (Shortest Path Distance) Queries. One of the most important research topics related to our work is the reachability (shortest path distance) between two nodes over a network. Existing approaches to reachability checking trade-off between query efficiency and indexing cost, and thus can be classified into three categories: online search, transitive closure, and 2-hop labeling.

To determine reachability between two nodes u and v, the online search approach traverses from u to v using breadth- or depth-first search over the network G. This incurs $O(|E|)$ cost of query processing time, where $|E|$ is the number of edges contained in G. Lots of efforts [17, 18, 19] have been devoted to pruning the search space of BFS or DFS by pre-computing certain auxiliary information on G. For example, GRAIL [19] assigns each node in G with an interval such that u cannot reach v if u's interval does not fully contain v's interval. Given a reachability query from u to v, the algorithm starts DFS from the source node u and avoids visiting nodes whose interval does not cover that of the terminal node v, since they can never reach v. In spite of these pruning strategies, the online search approach still takes much more time than methods in the other two categories, making it inapplicable to real-time applications where query efficiency is a major concern.

The transitive closure approach [20, 21, 22, 23, 24, 25, 26], on the contrary, answers reachability queries within constant time. For each node u in G, it pre-calculates and maintains on disk a collection of all nodes that u can reach, namely a *transitive closure*. In this way, the algorithm determines reachability from u to v simply by checking whether (u, v) exists in the transitive closure matrix of the network. Obviously, the transitive closure approach is quite efficient, but it results in significant pre-computation and high storage consumption, and hence cannot easily scale to large networks. However, such an approach is still useful in situations where query efficiency is much more critical than storage consumption, as long as efficient algorithms can be proposed to reduce pre-computation.

To the best of our knowledge, 2-hop labeling approach [27, 28, 29, 30, 31, 32, 33, 34] is the most widely-adopted indexing scheme for reachability queries on large networks. It targets on designing indexes to reduce the space consumption of transitive closure and at the same time answer reachability queries efficiently. Typically, a *2-hop cover* of a network $G = (V, E)$ consists of two label sets $\{L_{in}(v)\}_{v \in V}$ and $\{L_{out}(v)\}_{v \in V}$, where $L_{in}(v)$ (resp. $L_{out}(v)$) represents the collection of nodes that can reach v (resp. nodes reachable from v). For any two nodes u and v, u can reach v if and only if $L_{out}(u) \cap L_{in}(v) \neq \emptyset$. Therefore, a reachability query can be answered within linear time by checking the intersection of the source node's out-label set and the terminal node's in-label set. Obviously, the 2-hop labeling approach also provides satisfying query efficiency, and thus can be regarded as a wonderful alternative for the transitive closure method when storage is limited.

3. PROBLEM STATEMENT

In this section, we briefly introduce some basic concepts and notations employed in the remaining of the paper, as well as a formal definition of entity linking. After that, we present an overview of our framework to entity linking. Table 1 summarizes major notations used in the paper.

Table 1: Summary of notations.

Notation	Definition
m	entity mention
e	entity
$\mathbb{E}_m$	candidate entity set of mention m
d	tweet with timestamp $d.t$ and author $d.u$
$\mathbb{D}_e$	tweets linked to entity e
$\mathbb{U}_e$	community associated with entity e
$Inf(u, \mathbb{U}_e)$	user u's influence in community associated with entity e
$R(u, v)$	weighted reachability between users u and v
$S_{in}(u, e)$	user u's interest in entity e
$S_p(e)$	popularity score of entity e
$S_r(e)$	recency score of entity e

3.1 Preliminary and Problem Definition

DEFINITION 1 (ENTITY). *An entity e corresponds to a real-world object which exists in itself.*

DEFINITION 2 (MENTION). *A mention m is a sequence of words extracted from text and referring to some entities.*

Mentions are ambiguous while entities are unique, meaning that there exists many-to-many correspondence between mentions and entities. Specifically, A mention can refer to multiple entities. In Fig. 1, "Jordan" is a mention of *Jordan (country)*, *Air Jordan*, *Michael Jordan (basketball)*, and *Michael Jordan (machine learning expert)* as well. Meanwhile, an entity might have multiple mentions corresponding to it. For example, both "NYC" and "The Big Apple" can refer to the entity *New York City*. To resolve such a many-to-many correspondence between mentions and entities, entity linking is required.

DEFINITION 3 (**ENTITY LINKING**). *Entity linking is the task of determining the most probable entity e^* for a mention m.*

A typical strategy adopted by existing work on entity linking [3][4][5][6][7][8][9][10][11][12][2][13] includes two steps:

1. Candidate Generation. Given a mention m, a candidate entity set $\mathbb{E}_m$ is generated where each $e_i \in \mathbb{E}_m$ is a possible entity that m might refer to. In Fig. 1, the candidate entity set of "Jordan" is $\mathbb{E}_{Jordan}$ ={*Jordan (country)*, *Air Jordan*, *Michael Jordan (basketball)*, *Michael Jordan (machine learning expert)*};

2. Scoring and Ranking. For each entity in the candidate entity set $e_i \in \mathbb{E}_m$, a score $S(e_i)$ is calculated based on features which are application-dependent sometimes. After that, the most probable entity e^* is determined directly according to the ranking of scores.

Our focus in this work is how to calculate scores of entities in the candidate entity set, in order to achieve a reasonable ranking of candidate entities. we will briefly describe our approach to candidate generation in the next subsection. One thing to note is that a knowledgebase is required to enable machines to generate candidate entities automatically.

DEFINITION 4 (KNOWLEDGEBASE). *A knowledgebase $\mathbb{K}$ is a collection of mentions and entities along with mappings between them.*

DEFINITION 5 (COMPLEMENTED KNOWLEDGEBASE). *A complemented knowledgebase $\mathbb{K}'$ is a knowledgebase where each entity is associated with a list of tweets mentioning that entity along with their timestamps and authors, as illustrated in Fig. 1.*

We have discussed in Section 1.1 that the most important features for entity linking in Web documents are contextual similarity and topical coherence. Wikipedia, which is also the knowledgebase adopted in our work, is sufficient to provide these statistical information. However, queries and tweets are naturally short and informal, making both contextual similarity and topical coherence inapplicable. One of the most notable contributions of our work is that we propose new and effective features, namely user interest by social interactions, entity popularity and entity recency, which we believe are better options for entity linking in queries and tweets. In order to calculate these features, the original Wikipedia requires to be complemented with additional knowledge. Specifically, we pre-process a collection of tweets using current state-of-the-art approach to entity linking [2], and complement each entity in Wikipedia with a list of tweets mentioning that entity along with their timestamps and authors. We denote the collection of tweets linked to entity e as $\mathbb{D}_e$. Fig. 1 illustrates part of the complemented knowledgebase, where tweets associated with entities e_1 = *Michael Jordan (basketball)* and e_2 = *Michael Jordan (machine learning expert)* are $\mathbb{D}_{e_1} = \{d_5, d_{13}, d_{15}\}$ and $\mathbb{D}_{e_2} = \{d_{14}\}$ respectively. In the rest of this paper, we will use *Knowledgebase* and *Complemented Knowledgebase* interchangeably whenever context is clear.

We also mentioned in Section 1.2 that social interaction between users (i.e., followee-follower relationship) is a better choice to learn user interest, compared with tweet history. Although topics of users' broadcastings range from daily activities to breaking news and it is relatively difficult to detect user interest from such a diverse stream of tweets, users subscribe to others largely based on what they care. Therefore, a user's interest in an entity can be modeled as her interest in following users talking about that entity.

DEFINITION 6 (COMMUNITY). *A community $\mathbb{U}_e$ is a collection of users tweeting about a specific entity e, namely $\mathbb{U}_e = \{d.u | d \in \mathbb{D}_e\}$ where $d \in \mathbb{D}_e$ represents that tweet d is linked to entity e in our complemented knowledgebase.*

In Fig. 1, the communities associated with entities e_1 = *Michael Jordan (basketball)* and e_2 = *Michael Jordan (machine learning expert)* are $\mathbb{U}_{e_1} = \{u_3, u_7, u_9\}$ and $\mathbb{U}_{e_2} = \{u_6\}$ respectively.

3.2 Framework Overview

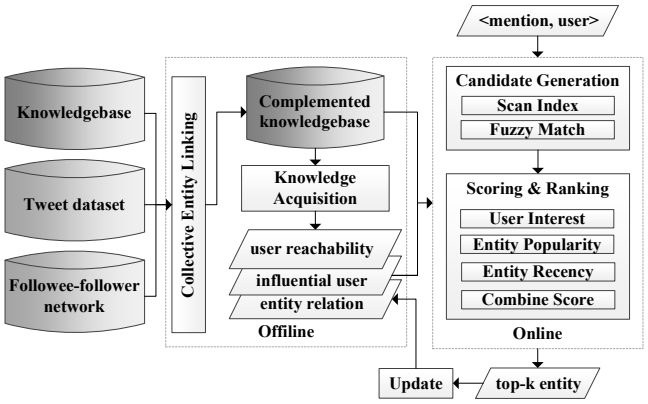


Figure 2: Framework overview.

Fig. 2 illustrates the framework of our prototype system for entity linking, which consists of two parts: offline knowledge acquisition and online inference. The workflow of our system is as below:

3.2.1 Offline Knowledge Acquisition

This module complements the original knowledgebase (Wikipedia) by pre-processing a set of tweets using current state-of-the-art approach to entity linking [2], and associating each entity in the knowledgebase with a list of tweets mentioning that entity along with their timestamps and authors, as demonstrated in Fig. 1. After that, it harvests essential knowledge which will be used during online inference.

Collective entity linking. [2] conducts entity linking in a batch manner. It assumes that each microblog user has an underlying interest distribution over named entities. Therefore, the intra-tweet entity relatedness and the inter-tweet user interest information can be integrated into a unified graph-based framework for collective entity linking. Specifically, given a set of entity mentions derived from all tweets posted by a user, it first constructs a graph between candidate entities where edge weights are calculated using the Wikipedia Link-based Measure (WLM) described in [35]. Then it estimates an initial interest score for each entity by considering its popularity, context similarity, as well as topical coherence, and employs a PageRank-like algorithm to propagate user interest from one entity to another. Finally, entities with the largest interest scores are regarded as the entity linking results.

Knowledge acquisition. We collect three types of knowledge during this process, namely weighted reachability between users,

collections of most influential users broadcasting about each entity, and topical relatedness between entities. In this work, we determine a user's interest in an entity e by considering her interest in following the community broadcasting about e, namely $\mathbb{U}_e$. This requires a reachability checking between the target user and users in $\mathbb{U}_e$. In order to improve efficiency, we identify a set of most influential users for each community and check reachability only with these users. We describe the details of reachability checking and influential user detection in Sec. 4.1. Entity recency, which is aggregated from associated tweets, is also important for entity linking. We observe that recency can be propagated among related entities. Therefore, we pre-calculate topical relatedness between entities to facilitate recency propagation, as described in Sec. 4.2.

3.2.2 Online Inference

Given an entity mention along with its author, this module first generates a collection of candidate entities. Then for each candidate, it calculates a score which is a combination of user interest, entity popularity, and entity recency. Finally, it outputs a list of top-k entities according to the estimated scores. One thing to note is that our framework for entity linking can be easily designed to support personalized microblog search. Specifically, if the input entity mention comes from a keyword query, our system will collect tweets linked to the top-k entities from the complemented knowledgebase and regard them as answers to that query; but if the entity mention is derived from a tweet, our system will interactively consult users for the correctness of the top-k entities, and link the tweet as well as its timestamp and author to each of the correct entities in the knowledgebase, and update existing knowledge such as user influences in the corresponding communities.

Candidate generation. Since this is not the focus of our work, we simply adopt approaches proposed in previous work [9, 36]. In particular, given an entity mention m, we generate a set of candidate entities $\mathbb{E}_m$ which might by referred to by m. To ensure flexibility, our knowledgebase contains various surface forms of named entities, such as name variations, nicknames, abbreviations, and so forth, which are derived from Wikepedia's *entity pages*, *redirect pages*, *disambiguation pages* and *hyperlinks in Wikipedia articles* (details described in [9]). For example in Fig. 1, if the entity mention m matches knowledgebase entry "Jordan", its candidate entity set will be $\mathbb{E}_m$ ={*Jordan (country)*, *Air Jordan*, *Michael Jordan (basketball)*, *Michael Jordan (machine learning expert)*}. As we know, queries and tweets are full of misspellings. Therefore, we build a segment-based index on knowledgebase entries and adopt fuzzy matching to obtain candidate entities based on *edit distance* similarity (details described in [36]).

Scoring and ranking. The score of each candidate entity in $\mathbb{E}_m$ is a combination of user interest, entity popularity and entity recency, namely:

$$S(e) = \alpha \cdot S_{in}(u, e) + \beta \cdot S_p(e) + \gamma \cdot S_r(e) \quad (1)$$

In Eq. 1, $\alpha + \beta + \gamma = 1$ are parameters indicating the importance of each feature for entity linking, which can be manually set based on experience or learned from labeled data. $S_{in}(u, e)$ measures user u's interest in entity e, and is calculated through reachability checking between u and the most influential users in e's community $\mathbb{U}_e$. We describe the details in Sec. 4.1. $S_r(e)$ is the recency score of entity e, which will be discussed in Sec. 4.2. $S_p(e)$ is the popularity score of entity e, and is calculated as follows:

$$S_p(e) = \frac{count(e)}{\sum_{e_i \in \mathbb{E}_m} count(e_i)} \quad (2)$$

where $count(e)$ is the number of tweets linked to entity e in the knowledgebase, and $\mathbb{E}_m$ is the candidate entity set of mention m.

4. METHODOLOGY

In this section, we describe some technical details in our framework for entity linking, including estimation of user interest by social interactions, as well as calculation and propagation of entity recency.

4.1 User Interest by Social Interactions

Traditional approaches to modeling user interest in microblogging sites are basically content-based, by assuming that users' interest might be scattered in their broadcastings. However, the topics of users' tweets usually range from daily activities to breaking news, making it difficult to detect user interest from such a diverse message stream. Furthermore, a large amount of microblogging users are information seekers who tweet rarely, which also enhances the difficulty to estimate their interest.

Fortunately, besides being content generators, microblogging users are also social network members, and they subscribe to others largely based on what they care. Therefore, we resort to social interactions between users (i.e., the followee-follower relationship) in this work, to avoid the difficulty and inaccuracy of modeling users' interest from their postings. In particular, we model a user's interest in an entity as her interest in following the community (i.e., the collection of users) broadcasting about that entity.

4.1.1 Weighted Reachability Checking

A straightforward way to measure a user's interest in a community is to calculate the number of users she subscribes to in that community, or in other words the proportion of overlapping between the community and her friend circle (i.e., followees). However, this naive method ignores users' interest in people several hops away. Such an indirect interest is also the motivation of friend recommendation in microblogging sites. Therefore, a better approach is to consider average reachability from the target user to people in the community through the directed followee-follower network.

$$S_{in}(u, e) = S_{in}(u, \mathbb{U}_e) = \frac{\sum_{v \in \mathbb{U}_e} R(u, v)}{|\mathbb{U}_e|} \quad (3)$$

In Eq. 3, $S_{in}(u, e)$ denotes user u's interest in entity e, which is transformed into u's interest in the community associated with entity e in our knowledgebase, namely $S_{in}(u, \mathbb{U}_e)$. And $R(u, v)$ denotes reachability from user u to user v.

However, Conventional reachability only considers connectedness between two nodes over a network, which is different from interestedness. The *small-world* theory guarantees that users in Twitter can be reachable within 4.12 steps in average ([16]), but it is obviously impossible for normal users to be interested in every other user. In order to achieve a meaningful measurement of users' interest in subscribing to other users, **reachability should be weighted**. We propose following heuristics:

- The shorter the distance from users u to v in the followee-follower network, the higher the probability of u following v;
- The larger the number of u's followees who participate in a shortest path from u to v, the higher the probability of u following v.

The second heuristic formulates strength of connection between users in the followee-follower network. Take 2-hop relationship

as an example. If most of u's followees subscribe to v, then u is probably interested in following v. Based on these heuristics, we define weighted reachability from u to v as follows:

$$R(u,v) = \frac{1}{d_{uv}} \cdot \frac{|F_{uv}|}{|F_u|} \quad (4)$$

In Eq. 4, F_u denotes the collection of users that u subscribes to (i.e., u's followees), and F_{uv} represents u's followees participating in at least one shortest path from u to v. Formally, $F_{uv} = \{t | t \in F_u \wedge t \in u \rightsquigarrow^* v\}$, where $u \rightsquigarrow^* v$ is the set of shortest paths from u to v whose length (i.e., number of edges in-between) is denoted as d_{uv}.

Extended Transitive Closure Framework

Since query efficiency is a major concern when conducting entity linking for personalized microblog search, we first assume access to unlimited storage resource and extend the transitive closure approach [20, 21, 22, 23, 24, 25, 26] for weighted reachability checking. Our focus here is to design algorithms to reduce time cost for offline pre-computation.

Obviously, a traverse over the entire followee-follower network is required to calculate weighted reachability between each pair of users. This is extremely time-consuming and thus unacceptable, even if reachability is calculated offline. In this work, we design an **incremental algorithm** to improve efficiency of reachability checking, as illustrated in Algo. 1.

Algorithm 1 Incremental algorithm for extended transitive closure

Input:
$G = (V, E)$ /*followee-follower network*/
H /*maximum number of hops*/
Output:
R /*weighted reachability matrix*/
1: Initialize each element of $R_{|V|\cdot|V|}$ as 0
2: **for** each $e = (u, v) \in E$ **do**
3: $R(u, v) = 1$
4: **end for**
5: **for** len=2:H **do**
6: **for** each $u \in V$ **do**
7: $T = \{t | R(u, t) = 1\}$ /*followees of u*/
8: **for** each $t \in T$ **do**
9: $V_t = \{v | R(t, v) \neq 0\}$ /*reachable from t in $len - 1$ hops*/
10: **end for**
11: $V = \langle v, n_v \rangle$ /*n_v is the number of V_t containing v*/
12: **for** each $v \in V$ **do**
13: **if** $R(u, v) = 0$ **then**
14: $R(u, v) = \frac{1}{len} \cdot \frac{n_v}{|T|}$ /*no shorter path from u to v, then update weighted reachability*/
15: **end if**
16: **end for**
17: **end for**
18: **end for**

Our incremental algorithm works by increasing the number of hops (i.e., the length of shortest path) one at a time. Given the followee-follower network $G = (V, E)$, we construct a matrix $R_{|V|\cdot|V|}$ to maintain weighted reachability between each pair of users. We traverse the network once and set reachability between connected users as 1 (lines 2-4). For each user u, her followees can be easily obtained by searching for elements in the reachability matrix $R(u, *)$ whose value is 1 (line 7). The key technique in this incremental algorithm lies in how to determine whether a followee of u (say t) participates in a shortest path from u to v, by leveraging existing knowledge about reachability.

Theorem 1. *There exists a len-hop shortest path from u to v which passes through u' followee t only if the shortest path distance from t to v is len − 1 hops.*

According to Theorem 1, we can determine whether to insert t into u's followee set participating in the *len*-hop shortest path from u to v simply by checking whether t can reach v within $len - 1$ hops (or whether $R(t, v) \neq 0$ after the $(len - 1)$-th iteration). We collect such followees, and update the weighted reachability matrix using Eq. 4 (lines 8-16).

Compared with the naive method which traverses the followee-follower network for reachability checking between each pair of users, our incremental algorithm directly or indirectly scans the network only H (i.e., maximum number of hops) times. More formally, the naive method needs to calculate pairwise weighted reachability for $O(|V|^2)$ times, where $|V|$ is the number of users in the followee-follower network. For each reachability checking, it uses a breadth-first search which traverses $O(|E|) = O(|V|^2)$ users. Altogether, the time complexity of the naive method is $O(|V|^4)$. Our incremental algorithm, on the other hand, can leverage existing information about reachability and thus achieves $O(H \cdot |V|^2)$ time complexity. According to a previous study on Twitter's network structure, users can be connected within 4.12 steps in average [16]. In other words, H is guaranteed to be a small value.

Extended 2-Hop Cover Framework

The transitive closure approach requires $O(|V|^2)$ space consumption, making it inapplicable when storage resource is limited. Recent work mostly targets on designing indexes to reduce the space consumption of transitive closure and at the same time answer reachability queries efficiently. We believe the 2-hop labeling approach [27, 28, 29, 30, 31, 32, 33, 34] is the most efficient and widely-used indexing method for reachability queries on large graphs. Therefore, we also consider extending the 2-hop cover framework in this work, to support weighted reachability queries when storage is limited.

Typically, a *2-hop cover* of a graph $G = (V, E)$ consists of two label sets $\{L_{in}(v)\}_{v \in V}$ and $\{L_{out}(v)\}_{v \in V}$, where $L_{in}(v)$ (resp. $L_{out}(v)$) represents the collection of nodes that can reach v (resp. nodes that can be reached from v). Given a reachability query $Query(s, t, L_{in}, L_{out})$ from node s to node t, the algorithm first retrieves the out-label set of s and the in-label set of t, and then calculates the intersection of these two label sets:

$$Intersect(s, t) = (L_{out}(s) \cup \{s\}) \cap (L_{in}(t) \cup \{t\})$$

If the intersection set $Intersect(s, t)$ is empty, then the algorithm returns $FALSE$ to reachability query $Query(s, t, L_{in}, L_{out})$; otherwise, t is regarded to be reachable from s.

In Eq. 4, we define weighted reachability from node u to node v in terms of the shortest path distance between u and v, as well as the number of u's followees who participate in at least one shortest path. Therefore, an extension of the original 2-hop cover is required to incorporate these information and support weighted reachability queries. In particular, we define two label sets for each node $v \in V$:

$$L_{in}(v) = \{(s, d_{sv})\}$$

$$L_{out}(v) = \{(t, d_{vt}, F_{vt})\}$$

where s in $L_{in}(v)$ (resp. t in $L_{out}(v)$) is a node that can reach v (resp. a node reachable from v), and d_{sv} (resp. d_{vt}) is the shortest path distance from s to v (resp. from v to t). F_{vt} in $L_{out}(v)$ is a collection of v's followees participating in the shortest path from v to t, namely $F_{vt} = \{u | u \in F_v \wedge u \in v \rightsquigarrow^* t\}$.

In order to calculate weighted reachability from users s to t (i.e., $R(s, t)$), we issue a weighted reachability query $Query(s, t, L_{in}, L_{out})$

to retrieve information required in Eq. 4 (i.e., d_{st} and F_{st}):

$$(d_{st}, F_{st}) = Query(s, t, L_{in}, L_{out}) = \begin{cases} d_{st} = \min\{d_{sv} + d_{vt} | (v, d_{sv}, F_{sv}) \in L_{out}(s) \wedge (v, d_{vt}) \in L_{in}(t)\} \\ F_{st} = \bigcup_{d_{sv}+d_{vt}=d_{st}} F_{sv} \end{cases} \quad (5)$$

Here, d_{st} is the shortest path distance from nodes s to t, and $d_{st} = \infty$ when s cannot reach t within H (i.e., predefined maximum number of hops) steps. F_{st} is the collection of followees that participate in the shortest path from s to t.

Theorem 2. *If the shortest path from s to v passes through s's followee f, and v lies in the shortest path from s to t, then f participates in the shortest path from s to t.*

According to Theorem 2, we calculate F_{st} in Eq. 5 by aggregating F_{sv} where v lies in the shortest path from s to t. For example, assume that $L_{out}(s) = \{(v_1, 2, \{f_1, f_2, f_3\}), (v_2, 3, \{f_2, f_4\})\}$ and $L_{in}(t) = \{(v_1, 2), (v_2, 1)\}$, then the shortest path distance from s to t is obviously $d_{st} = 4$ and s's followees participating in its shortest path to t is $F_{st} = \{f_1, f_2, f_3, f_4\}$.

Algorithm 2 Index construction for extended 2-hop labeling

Input:
$G = (V, E)$ /*followee-follower network*/
H /*maximum number of hops*/
Output:
$L_{in}^{|V|}$ and $L_{out}^{|V|}$ /*2-hop labels of G*/
1: Sort V by their degrees in descending order
2: **for** k=1:|V| **do**
3: $\forall v \in V, L_{in}^k(v) = L_{in}^{k-1}(v)$ and $L_{out}^k(v) = L_{out}^{k-1}(v)$
4: /*Conduct backward BFS to find all nodes that can reach v_k, and update L_{out}^k's distances and followee sets when necessary*/
5: Q = an empty queue; Enqueue $\langle v_k, 0\rangle$ onto Q
6: **while** Q is not empty **do**
7: Dequeue $\langle u, len\rangle$ from Q
8: $len = len + 1$
9: **for all** $s \in N_{in}(u)$ **do**
10: $(d_{sv_k}, F_{sv_k}) = Query(s, v_k, L_{in}^{k-1}, L_{out}^{k-1})$
11: **if** $len < d_{sv_k}$ **then**
12: /*If distance is shortened, then update L_{out}^k and add the current node onto the queue*/
13: **if** $v_k \in L_{out}^k(s)$ **then**
14: Remove $(v_k, d_{sv_k}, F_{sv_k})$ from $L_{out}^k(s)$
15: **end if**
16: $L_{out}^k(s) = L_{out}^k(s) \cup \{(v_k, len, \{u\})\}$
17: **if** $len < H \wedge s \notin Q$ **then**
18: Enqueue $\langle s, len\rangle$ onto Q
19: **end if**
20: **else if** $len = d_{sv_k} \wedge u \notin F_{sv_k}$ **then**
21: /*If a new shortest path is found, then update followee set*/
22: **if** $v_k \in L_{out}^k(s)$ **then**
23: $L_{out}^k(s).F_{sv_k} = L_{out}^k(s).F_{sv_k} \cup \{u\}$
24: **else**
25: $L_{out}^k(s) = L_{out}^k(s) \cup \{(v_k, len, \{u\})\}$
26: **end if**
27: **end if**
28: **end for**
29: **end while**
30: Conduct forward BFS to find all nodes that v_k can reach, and update L_{in}^k only when distance is shortened
31: **end for**

Our offline indexing algorithm for the extended 2-hop labeling framework is depicted in Algo. 2. Inspired by the Pruned Landmark Labeling (PLL, [32, 33]) approach, we sort nodes in $G = (V, E)$ by their degrees in descending order (line 1), and construct 2-hop labels L_{in}^k (resp. L_{out}^k) from L_{in}^{k-1} (resp. L_{out}^{k-1}) using information obtained by pruned forward (resp. backward) BFS from the node v_k (lines 2-31). Specifically, suppose that we are handling a node t with distance len during the forward BFS from v_k. We issue a weighted reachability query from v_k to t, namely $(d_{v_kt}, F_{v_kt}) = Query(v_k, t, L_{in}^{k-1}, L_{out}^{k-1})$. If $len < d_{v_kt}$ (i.e., we find a shorter path from v_k to t), then we add (v_k, len) to $L_{in}^k(t)$ (i.e., $L_{in}^k(t) = L_{in}^k(t) \cup (v_k, len)$) and insert t into a queue for further BFS. One thing to note is that the index updating strategy for L_{out} is slightly different. Since $L_{out}^k(s)$ records both the shortest path distance from s to v_k and the collection of s's followees participating in the shortest path, special attention is required to maintain the followee set when updating $L_{out}^k(s)$ (lines 5-29). Assume that we are visiting a node s with distance len during the backward BFS from node v_k, and the shortest path distance (resp. followee set) obtained by a weighted reachability query from s to v_k is d_{sv_k} (resp F_{sv_k}). We consider two situations here which trigger updates of $L_{out}^k(s)$: 1) $len < d_{sv_k}$ (lines 11-19). Like before, a shorter path from s to v_k (i.e., $s \rightarrow u \rightsquigarrow v_k$) is detected in this situation. We add $(v_k, len, \{u\})$ to $L_{out}^k(s)$ where u is the followee of s in the len-hop path from s to v_k, and insert s into a queue for further BFS. Since we traverse nodes using a backward BFS, s's followees u can be directly recorded during this process. 2) $len = d_{sv_k}$ (lines 20-27). A new len-hop shortest path from s to v_k (i.e., $s \rightarrow u \rightsquigarrow v_k$) is detected in this situation. We need to check whether the information that s's followee u participates in a shortest-path from s to v_k has been encoded in existing 2-hop labels. If not, we insert u into the followee set, namely $L_{out}^k(s).F_{sv_k} = L_{out}^k(s).F_{sv_k} \cup \{u\}$. Since the shortest path distance from s to v_k stays unchanged, which means the distances from s's ancestors to v_k passing through s must also be the same, we do not need to add s into the queue for further BFS.

We evaluate the performance of the extended transitive closure framework and the extended 2-hop cover framework for weighted reachability queries in Sec. 5, in terms of query efficiency, space consumption, as well as pre-computation time.

4.1.2 Detecting influential users

As depicted in Eq. 3, measuring a user u's interest in an entity e requires reachability checking between u and other users in the community associated with e. Although these information can be obtained directly from the pre-computed weighted reachability matrix, aggregating such a great amount of reachability during online inference is still a waste of time. Moreover, it is obviously inappropriate to treat every user as equal. Different people have different influences in a community, and a user's interest in influential people has more contribution to her interest in the community. To improve both the accuracy and the efficiency of user interest estimation, we propose to detect a collection of most influential users for each community and consider reachability only with those influential users.

Intuitively, a user is influential in a community associated with an entity if:

- She posts a large proportion of tweets linked to that entity;
- She is discriminative among candidate entities.

The second intuition guarantees that the influential user of an entity has a *distinct* interest in broadcasting about that entity, and such an interest *remains stable* over time. For example, a machine learning expert might also has a major interest in basketball, and will talk about both *Michael Jordan (basketball)* and *Michael Jordan (machine learning expert)* in her tweets. Hence, estimating target user u's interest in following such a user makes little contribution to discriminating candidate entities of the mention "Jordan". On the contrary, NBA's official account in Twitter (i.e., @NBAOfficial) hardly

broadcasts about *Jordan (country)*, *Air Jordan*, or *Michael Jordan (machine learning expert)*, making u's subscription to @NBAOfficial an important hint of her interest in basketball. In this case, mentions of "Jordan" in u's queries and tweets should probably be linked to *Michael Jordan (basketball)*.

Tfidf-Based Approach

One possible approach to identify influential users in a community considering the aforementioned intuitions is to adopt the tf-idf weighing strategy. Given the candidate entity set $\mathbb{E}_m$ of mention m, we can obtain documents and community associated with each candidate entity e from our complemented knowledgebase, namely $\mathbb{D}_e$ and $\mathbb{U}_e$ respectively. We use the following formula to measure the influence of a user u in community $\mathbb{U}_e$:

$$Inf(u, \mathbb{U}_e) = \frac{|\mathbb{D}_e^u|}{|\mathbb{D}_e|} \cdot \log \frac{|\mathbb{E}_m|}{|\mathbb{E}_m^u|} \tag{6}$$

In Equation 6, $\mathbb{D}_e^u = \{d | d \in \mathbb{D}_e \wedge d.u = u\}$ denotes the collection of tweets linked to entity e which are posted by user u. Hence, $\frac{|\mathbb{D}_e^u|}{|\mathbb{D}_e|}$ indicates u's enthusiasm in broadcasting about entity e. On the other hand, $\log \frac{|\mathbb{E}_m|}{|\mathbb{E}_m^u|}$ reflects u's ability to differentiate candidate entities (namely whether u's interest is unique or diversified among all candidate entities), where $\mathbb{E}_m^u = \{e_i | e_i \in \mathbb{E}_m \wedge \exists d \in \mathbb{D}_{e_i}, d.u = u\}$ denotes the set of candidate entities that user u has mentioned at least once in her tweets.

Entropy-Based Approach

In practice, it is possible that an influential user in a community (say @NBAOfficial) occasionally tweets about candidate entities of other communities (say *Air Jordan*). Such an incident posting should not cause huge impact on her influence in the original community. In this sense, the tfidf-based approach is inappropriate, since it penalizes a user's influence as long as she has broadcastings in many communities. A better approach is to consider the probability distribution of a user's tweets among candidate entities.

Entropy can be regarded as a quantitative description of the shape of a probability distribution, which maximizes when the distribution is uniform. Obviously, a user is discriminative among candidate entities if and only if the probability distribution of her tweets among these entities is biased, which in turn means a small entropy value. Therefore, we propose an entropy-based approach to estimate user influence in a community:

$$Inf(u, \mathbb{U}_e) = \frac{|\mathbb{D}_e^u|}{|\mathbb{D}_e|} \cdot \frac{1}{entropy(u, \mathbb{E}_m)} \tag{7}$$

As before, $\frac{|\mathbb{D}_e^u|}{|\mathbb{D}_e|}$ in Eq. 7 reflects user u's contribution of tweets to the community associated with entity e. $entropy(u, \mathbb{E}_m)$ measures u's ability to differentiate candidate entities, which is calculated as below:

$$entropy(u, \mathbb{E}_m) = -\sum_{e_i \in \mathbb{E}_m} p(u, e_i) \log p(u, e_i)$$

$$p(u, e_i) = \frac{|\mathbb{D}_{e_i}^u|}{\sum_{e_k \in \mathbb{E}_m} |\mathbb{D}_{e_k}^u|}$$

Here, $p(u, e_i)$ is the probability of user u mentioning entity e_i in her tweets, as compared with other entities in the candidate entity set $\mathbb{E}_m$. And $\mathbb{D}_{e_i}^u = \{d | d \in \mathbb{D}_{e_i} \wedge d.u = u\}$ is the collection of tweets written by user u about entity e_i.

We will evaluate the effectiveness of both tfidf-based and entropy-based approaches to user influence estimation in Sec. 5. Given influence scores of users in a community $\mathbb{U}_e$, the collection of most influential users in that community can be identified by a simple ranking:

$$\mathbb{U}_e^* = \arg\max_u Inf(u, \mathbb{U}_e)$$

After that, we can estimate a user's interest in entity e by her interest in following those influential users, which is in turn estimated through weighted reachability checking. Formally,

$$S_{in}(u, e) = S_{in}(u, \mathbb{U}_e^*) = \frac{\sum_{v \in \mathbb{U}_e^*} R(u, v)}{|\mathbb{U}_e^*|} \tag{8}$$

where $\mathbb{U}_e^*$ is the set of influential users associated with entity e, and $R(u, v)$ is the weighted reachability from users u to v.

4.2 Entity Recency

As discussed in Sec. 1, recency is important to entity linking especially when it is conducted to support personalized microblog search. Entity recency can be obtained indirectly by gathering temporal information of tweets talking about that entity. However, linking a single newly-generated tweet to an entity does not necessarily mean a refresh of entity recency. Instead, entity recency can be identified only when a burst of tweets about that entity occurs within short time. In this work, we adopt a simple but effective approach to measure entity recency - **sliding window**.

Given a time window τ, we define *recent tweets* of entity e as those linked to e and published within τ till the current time. If the number of recent tweets exceeds a predefined threshold θ_1, we believe that there has been a burst of attention on entity e, which implies the freshness of e. Since entity recency is proposed as a feature to support entity linking, namely estimating the probability of mapping an entity mention to a specific entity in the candidate entity set, we normalize recency of entity e over all candidate entities:

$$S_r(e) = \begin{cases} \frac{|\mathbb{D}_e^\tau|}{\sum_{e_i \in \mathbb{E}_m} |\mathbb{D}_{e_i}^\tau|} & |\mathbb{D}_e^\tau| \geq \theta_1 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

In Eq. 9, $\mathbb{D}_e^\tau = \{d | d \in \mathbb{D}_e \wedge d.t \geq now - \tau\}$ denotes recent tweets of entity e concerning time window τ.

Besides a burst of tweets mentioning entity e, recency of e can also be indirectly signified by that of related entities. For example, the recency of *Chicago Bulls* and *NBA* (National Basketball Association) enhances that of *Michael Jordan (basketball)*. Similarly, increasing amount of tweets about *ICML* (International Conference on Machine Learning) implies more attention on machine learning experts like Michael Jordan *(machine learning expert)*. Therefore, we propose a **recency propagation model** to incorporate mutual reinforcement of recency between related entities.

We adopt the Wikipedia Link-based Measure (WLM) described in [4] to calculate topical relatedness between entities. WLM assumes that two Wikipedia articles are topically related if the number of Wikipedia articles linked to both of them is large. Formally, given two entities e_i and e_j, we define topical relatedness between them as follows:

$$Rel(e_i, e_j) = 1 - \frac{\log(\max(|A_{e_i}|, |A_{e_j}|)) - \log(|A_{e_i} \cap A_{e_j}|)}{\log(|A|) - \log(\min(|A_{e_i}|, |A_{e_j}|))} \tag{10}$$

where A is the entire set of Wikipedia articles, and A_e is the set of Wikipedia articles that link to Wikipedia article describing entity e. Obviously, the more topically related e_i and e_j are, the larger $Rel(e_i, e_j)$ will be.

Fig. 3 illustrates an example of the recency propagation network whose nodes are entities in our knowledgebase. We propose several heuristics on constructing the network:

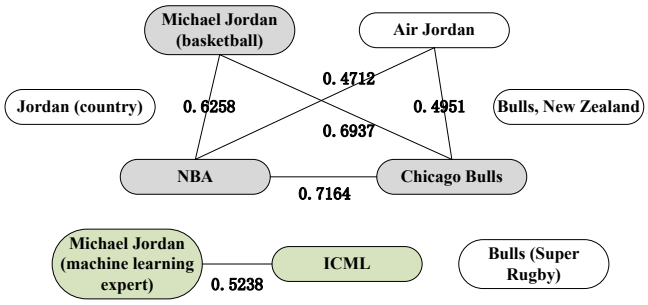


Figure 3: Example of recency propagation network.

- Since recency is used to find the best mention-entity mapping, it should not be propagated between candidate entities of the same entity mention;
- If two entities are more interdependent or topically related with each other, recency should be propagated between them in a larger extent;
- Only highly related entities can reinforce recency of each other. This avoids extensive recency diffusion to slightly related entities.

The first heuristic indicates that candidate entities of the same mention, such as *Jordan (country)*, *Air Jordan*, *Michael Jordan (basketball)*, and *Michael Jordan (machine learning expert)*, should not be connected in the recency propagation network; The second heuristic implies that edge weight should be defined as the topical relatedness between corresponding entities; And according to the third heuristic, we remove edges whose weights are smaller than a predefined threshold θ_2 from the entity relatedness network, and conduct a *Graph-Cut* to find clusters of strongly connected entities. We add edges between entities in such clusters into the recency propagation network.

Given the recency propagation network $G = (V, E)$, our propagation algorithm runs as follows: First, we initialize recency of entities in our knowledgebase based on their recent tweets, namely Eq. 9. This results in an initial recency vector denoted as $\vec{S_r}^0$. Then we reinforce recency between entities iteratively using a PageRank-like algorithm. Edges in the recency propagation network constitute a path to propagate recency from one entity to another. We normalize edge weights as below, to formulate the probability of recency reinforcing:

$$P(e_i, e_j) = \frac{w(e_i, e_j)}{\sum_{(e_i,e_k)\in E} w(e_i, e_k)}$$

Here, $w(e_i, e_j)$ is the edge weight (i.e., topical relatedness) between entities e_i and e_j. Given the initial recency vector $\vec{S_r}^0$ and the recency propagation matrix P, we can get the final recency vector using the following iterative process:

$$\vec{S_r}^i = \lambda \cdot \vec{S_r}^0 + (1 - \lambda) \cdot P \cdot \vec{S_r}^{i-1} \quad (11)$$

In Eq. 11, λ is a parameter to trade-off between recency gathered from underlying tweets and that reinforced by other entities.

5. EXPERIMENT

We conducted extensive experiments to evaluate the effectiveness and efficiency of our framework for entity linking in microblogging services. All the algorithms were implemented in C#, and all the experiments were conducted on a server with 2.90GHz Intel Xeon E5-2690 CPU and 192GB memory.

5.1 Experimental Setting

5.1.1 Wikipedia dataset

We downloaded the July 2014 version of English Wikipedia, which was used to construct our original knowledgebase defined in Sec. 3. The Wikipedia dump contains 6.3 million redirect pages, 0.2 million disambiguation pages, and 19.2 million entity pages. 380 million hyperlinks are established between entity pages, and from anchor texts associated with these hyperlinks we extracted 3.8 million nicknames of entities. Altogether, we obtained 29.3 million mentions (e.g., "Bulls", "Tony Allen", and "Jordan" in Fig. 1) and 19.2 million entities (e.g., *Chicago Bulls*, *Tony Allen (basketball)*, and *Michael Jordan (basketball)* in Fig. 1) together with mappings between them. We pre-calculated topical relatedness between entities using Wikipedia Link-base Measure (WLM, Eq. 10), which will be used in our recency propagation model, as well as the estimation of topical coherence considered in current state-of-the-art approaches to entity linking [14, 2]. We also harvested the popularity and the context of each entity from Wikipedia, to calculate intra-tweet features adopted in [14] and [2].

5.1.2 Twitter dataset

To guarantee that our knowledgebase has integrated as many entities mentioned in tweets as possible, the tweets we collect for experiments should be generated before the timestamp of Wikipedia dump. In particular, we used Twitter's API to download a collection of 29.3 million tweets which were published by 5 million users between September 2012 and February 2013. After conducting Named Entity Recognition (NER) using current start-of-the-art approaches to NER in tweets [37, 38], we observed that 12 million tweets in our dataset (40.96%) contain at least one named entity mention. This is slightly smaller that the 45.08% percentage found in previous work [2].

We conducted a collective entity linking [2] on part of the tweet dataset to complement our knowledgebase. Since [2] assumes that a user's interest is scattered amongst her broadcastings, it requires that the target user should have sufficient amount of historical tweets to infer her interest. Therefore, we extracted a set of active users with more than θ postings during this period, and used their tweets for the collective entity linking. We ranged θ from 10 to 90 incremented by 20, and obtained five tweet datasets: $\mathbb{D}10$, $\mathbb{D}30$, $\mathbb{D}50$, $\mathbb{D}70$, and $\mathbb{D}90$. In this work, we claim that one of the most notable superiorities of our framework is its ability to maintain accuracy of entity linking for information seekers who tweet rarely. To verify this, we randomly sampled 200 inactive users (with less than 10 broadcastings) and used their tweets for testing: $\mathbb{D}test$. Table 2 summarizes the statistics of these tweet datasets.

Table 2: Statistics of tweet datasets.

$\mathbb{D}10$		$\mathbb{D}30$		$\mathbb{D}50$	
#user	#tweet	#user	#tweet	#user	#tweet
311,835	6.76M	46,476	2.69M	16,015	1.56M
$\mathbb{D}70$		$\mathbb{D}90$		$\mathbb{D}test$	
#user	#tweet	#user	#tweet	#user	#tweet
7698	1.01M	4422	0.82M	200	649

5.1.3 Evaluation Method

We compared our approach to entity linking in tweets with two state-of-the-art methods, in terms of both accuracy and efficiency.

[14] conducts entity linking on-the-fly (i.e., tweet by tweet), by considering intra-tweet features such as entity popularity in Wikipedia, context similarity between mentions and entities, topical coherence within tweets, and so forth. [2] combines intra-tweet features with inter-tweet interest distributions, and thus achieves entity linking in a batch manner. We invited three colleagues to help with the labeling, by providing them with the entity linking results of each tweet along with its author, as well as some recent tweets containing the same entity mentions. Annotators were required to examine the friend network of the target user, her historical broadcastings and the recent tweets, to make their decisions. The final labels were based on majority voting. Table 3 lists default values of parameters used in this work.

Table 3: Default values of parameters.

importance of user interest, entity recency, and popularity		
α	β	γ
0.6	0.3	0.1
# recent tweet within time window		highly-related
τ	θ_1	θ_2
3 days	10	0.6

5.2 Experimental Results

5.2.1 *Effectiveness*

Fig. 4 (a) illustrates the accuracy of entity linking achieved by on-the-fly approach [14], collective approach [2], and our prototype system respectively. We present the percentage of correctly linked mentions, and that of tweets whose mentions are all correctly linked. As expected, the accuracy of mentions is always larger than that of tweets. We observe that the collective entity linking method performs better than the on-the-fly counterpart, since it considers users' interest distribution among their tweets, and combines inter-tweet features with traditional intra-tweet features to conduct entity linking. However, such an improvement is not as significant as that discovered in [2]. This is mainly because the collective method requires large amount of tweets to estimate a user's interest distribution. Whereas we focus on entity linking for information seekers with limited tweets in this work. Specifically, users in our test dataset only have 3.25 tweets in average, making the improvement of collective method insignificant. Our approach, on the other hand, achieves better performance by inferring user interest based on social interactions, and combining such interest with entity recency and entity popularity to detect the best linking.

An important prerequisite to our entity linking approach is the quality of the complemented knowledgebase. Therefore, we compared the accuracies of entity linking when complementing Wikipedia with different sizes of datasets. Fig. 4 (b) shows the result. We can see that with increasing amount of tweets appended to the original knowledgebase, the accuracy of entity linking improves gradually. This is consistent with our expectation. Nevertheless, we still observe a slight degradation of performance from $\mathbb{D}70$ to $\mathbb{D}50$, which is caused by mistakenly linked tweets when conducting collective entity linking on users with limited number of tweets. This is actually a trade-off between quality and coverage of the knowledgebase.

In our propotype system, we combine user interest, entity recency, and entity popularity to conduct entity linking. In Eq. 1, α, β, and γ reflect their relative contributions to the overall scoring. Table 4 illustrates the accuracies achieved by considering these features separately and all together. We can see that user interest is the most important feature to support personalized entity linking.

Table 4: Effectiveness of user interest, entity recency, and entity popularity for entity linking.

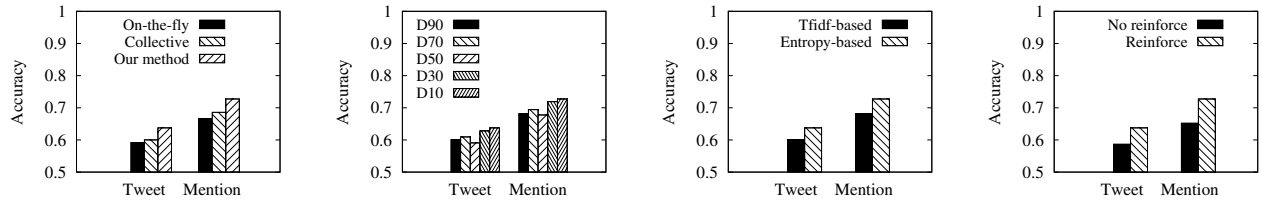
	$\alpha = 1$	$\beta = 1$	$\gamma = 1$	all features
tweet	0.6281	0.6000	0.5906	0.6375
mention	0.7190	0.6860	0.6777	0.7273

And the performance of user interest estimation using social interactions is better than that obtained using content-based approaches (i.e., 0.686 mention accuracy and 0.6 tweet accuracy achieved by collective method in Fig. 4 (a)), especially for information seekers. Entity recency performs better than entity popularity, since recency is a time-dependent feature which measures an entity' recent popularity. By combining all these features, our framework achieves the highest accuracy.

In this work, we propose new approaches to estimate user interest and entity recency respectively. We evaluated their effectiveness in our experiments. We measure a user's interest in an entity by examining her interest in following the community (i.e., set of users) tweeting about that entity, which is in turn formulated as the average weighted reachability between the target user and the most influential users in the community. We propose two methods to calculate user influence: tfidf-based and entropy-based. Fig. 4 (c) demonstrates their performance reflected by the accuracy of entity linking. We can see that the entropy-based approach performances better than the tfidf counterpart, since it allows for the case that an influential user in a community (e.g., @NBAOfficial) occasionally tweets about entities in other communities (e.g., *Air Jordan*). We propose a recency propagation model to estimate entity recency. Fig. 4 (d) shows the necessity and performance of recency reinforcing. As we have discussed before, the recency of *NBA* will increase that of *Michael Jordan (basketball)*. And similarly the recency of *ICML* might also indicates a burst of tweets about *Michael Jordan (machine learning expert)*. Therefore, the accuracy of entity linking which incorporates propagated recency outperforms that without recency reinforcement in Fig. 4 (d).

5.2.2 *Efficiency*

Entity linking is usually regarded as an online task, or an underlying step of many other text processing algorithms such as clustering and classification. This requires that it should be conducted within short time. Fig. 5 (a) illustrates the time requirement of our framework for entity linking, compared with current state-of-the-art methods. We present the average time required to link a single mention and a whole tweet respectively. As expected, existing on-the-fly approach is the most efficient, since it only considers intra-tweet features which can be calculated easily. The collective method incorporates inter-tweet features. It propagates user interest among entities extracted from all the tweets of a specific user, which is obviously time-consuming when the user has a large amount of tweets. However, since we focus on entity linking for information seekers in our experiments, the collective method only needs to handle a few mentions each time. In our test dataset, each user has 3.25 tweets and each tweet has 1.36 mentions in average, making the collective method to be extremely fast. Our approach, on the contrary, employs a PageRank-like algorithm to model recency reinforcement between related entities. Since a popular entity (e.g., *Michael Jordan (basketball)*) usually has a large number of related entities (e.g., *NBA*, *Chicago Bulls*, *Kobe Bryant*, etc.), the efficiency of our framework is incomparable with existing on-the-fly method. Nevertheless, our approach can still link entities of a tweet within 0.5 milliseconds by constraining recency propagation only between highly related entities, which we believe is



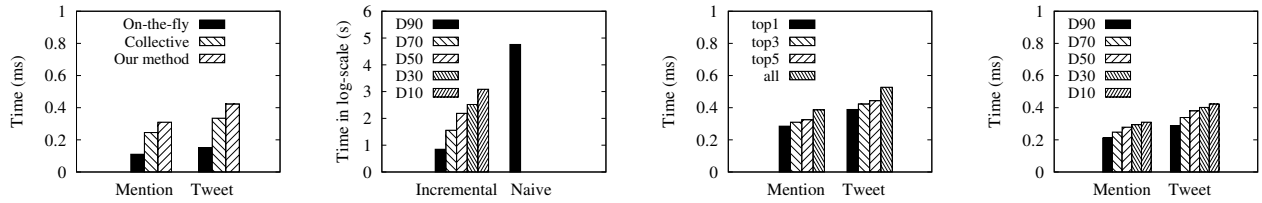
(a) effectiveness of our framework compared with state-of-the-art methods.

(b) effectiveness of different tweet datasets used for knowledgebase complementation.

(c) effectiveness of tfidf-based and entropy-based approach for recency propagation model.

(d) necessity and performance of user influence estimation.

Figure 4: Results of effectiveness evaluation.



(a) efficiency of our framework compared with state-of-the-art methods.

(b) efficiency of the naive and incremental algorithms to calculate reachability transitive closure.

(c) efficiency improved by considering the most influential users.

(d) efficiency of our framework when knowledgebase expands.

Figure 5: Results of efficiency evaluation.

sufficiently fast for most real-world applications. For example, Twitter's data showed that 200 million users send over 400 million tweets daily as of September 2013[4], which means 5000 postings are generated every second. Given that 40% of tweets contain at least one named entity mention, the entity linking framework needs to handle each tweet within 0.5 milliseconds. In this sense, our approaches are feasible for entity linking in real-time. Besides, our framework can be easily parallelized, since it conducts entity linking independently (i.e., tweet by tweet or even mention by mention) and requires no information from any other tweet. This can further improve the efficiency and applicability of our methods for online entity linking.

In this work, we measure a user's interest in an entity by calculating weighted reachability between the target user and users in the community associated with that entity. Since efficiency is the major concern in order to support online entity linking, we assume access to unlimited storage consumption, and extend the transitive closure framework for weighted reachability query. Naive approach to offline transitive closure construction needs to traverse the entire followee-follower network to calculate reachability for each pair of users, and thus is inefficient. We propose an incremental algorithm to improve the efficiency. Fig. 5 (b) shows the pre-computation time for transitive closure in log-scale. We omit results of index construction that cannot be finished within one day. From Fig. 5 (b) we can see that our incremental algorithm is extremely faster than the naive method. In particular, it takes less than 20 minutes to calculate all the pairwise weighted reachability in our largest dataset.

We also extend the 2-hop cover framework to support weighted reachability queries in case of limited storage resource, considering its ability to reduce space consumption of the transitive closure and at the same time answer reachability queries efficiently. We evaluated the performance of the extended transitive closure framework and the extended 2-hop cover framework on our datasets, in terms of query efficiency, index size, as well as pre-computation time. In addition to the filtered datasets adopted in previous experiments (i.e., $\mathbb{D}10$, $\mathbb{D}30$, $\mathbb{D}50$, $\mathbb{D}70$ and $\mathbb{D}90$), we also evaluated our approaches on the whole crawled dataset (i.e., $\mathbb{D}$ with 5 million users), as well as a public Twitter social network dataset[5] consisting of 11.3 million users. In order to evaluate query efficiency, we randomly sampled 1000 source nodes and another 1000 terminal nodes from each dataset, and generated 1,000,000 weighted reachability queries. We calculated the average query time to measure the efficiency of our approaches. Table 5 illustrates the statistics of these datasets and the performance of our indexing strategies for weighted reachability queries. As for the extended transitive closure framework, we only present the index construction time after applying our incremental algorithm. Results for index construction that cannot be finished within one day or that occupies excessive memory resource are ignored. We can see that the transitive closure approach answers reachability queries much faster than the 2-hop cover counterpart, at the cost of higher storage consumption and longer index construction time. On the other hand, the 2-hop cover framework, besides reducing index sizes dramatically, can still answer reachability queries efficiently, which makes it a wonderful alternative to handle large graphs (e.g., $\mathbb{D}$ and Twitter in Table 5) when storage resource is limited. Note that the index sizes, indexing time, and query efficiency of our approaches are slightly worse than those achieved in previous work on reachability queries. But this is mainly due to the fact that extra operations are required to maintain the followee sets in-between the shortest paths from nodes s to t, in order to calculate weighted reachability.

Although reachability can be retrieved directly after offline precalculation, estimating a user's average reachability with a community is still time-consuming, since the communities associated with popular entities are usually very large. Therefore, we propose to detect a collection of most influential users in a community, and check reachability only with those influential users. Fig. 5 (c) illustrates

[4]http://en.wikipedia.org/wiki/Twitter#Growth

[5]http://www.datatang.com/data/44253/

Table 5: Performance of the extended transitive closure framework and the extended 2-hop cover framework.

	#node	#edge	avg degree	max degree	indexing time		index size		query time	
					transitive closure	2-hop	transitive closure	2-hop	transitive closure	2-hop
$\mathbb{D}$90	4.6K	52.3K	22.6	0.7K	7s	2s	67MB	11MB	0.1μs	0.3μs
$\mathbb{D}$70	7.9K	123.07K	31.1	1.1K	36s	16s	154MB	31MB	0.1μs	0.4μs
$\mathbb{D}$50	16.2K	343.1K	42.3	1.9K	156s	86s	655MB	102MB	0.2μs	1.2μs
$\mathbb{D}$30	46.7K	899.1K	38.5	3.5K	328s	239s	1.8GB	271MB	0.1μs	0.8μs
$\mathbb{D}$10	312.1K	5.8M	34.3	6.8K	1217s	471s	8.7GB	1.4GB	0.2μs	1.5μs
$\mathbb{D}$	5.0M	71.4M	28.61	412.76K	-	1893s	-	16GB	-	4.7μs
Twitter	11.3M	85.3M	15.1	564.8K	-	3417s	-	38GB	-	4.2μs

the variation of time requirement when we increase the number of users to check reachability with. We observe an insignificant difference in Fig. 5 (c). This is mainly because that the tweet dataset we used for knowledgebase complementation in our experiments is quite small compared with the whole tweet dataset in Twitter. Hence, even the communities associated with popular entities contain only a small number of users. However, we can still find a trend of increasement of time when calculating reachability with more users.

At last, we examined the scalability of our framework for entity linking. From Fig. 5 (d), we can see that the time requirement of our system remains stable when we complement our knowledgebase with increasingly larger tweet dataset. The most time-consuming modules in our framework are reachability checking with community and the recency propagation model. However, after restricting reachability checking only with influential users and recency propagation only between highly related entities, both modules are insensitive to the number of tweets complemented to the original knowledgebase.

6. CONCLUSION

In this paper, we propose an on-the-fly approach to entity linking in tweets, which takes into consideration a combination of user interest, entity recency, and entity popularity. Unlike existing content-based methods, we measure user interest by social interactions, considering the difficulty and inaccuracy in inferring user interest from a diverse stream of tweets with highly diversified topics. We formulate a user's interest in an entity as her interest in subscribing to the community tweeting about that entity, which is calculated by weighted reachability between the target user and the most influential users in the community. We propose two approaches for user influence estimation within a community, namely tfidf-based approach and entropy-based approach. Effective indexing structures along with incremental algorithms are developed to reduce the time requirement for calculating weighted reachability. We adopt a simple but effective method - sliding window - to estimate an entity's initial recency, and propose a page-rank style recency propagation model to incorporate recency reinforced by other highly related entities. Empirical studies have demonstrated the superior effectiveness of our approach compared with state-of-the-art methods and the efficiency feasibility for online applications with real-time requirement.

Currently, we evaluate our approaches by providing annotators with top-k entities for each mention, and asking them to examine the correctness of the entity linking result, which is labour-intensive, time consuming and sometimes subjective. A better approach is to build an interface for our methods on Twitter, and invite tweet authors themselves to conduct the evaluation. We are currently working on this development. Furthermore, our framework for entity linking can be easily extended to support personalized microblog search. However, in order to build a robust search engine, the entire followee-follower network as well as an extremely huge collection of tweets are required. We are seeking for cooperations with companies and other research teams to achieve this goal.

Acknowledgement

This work was partially supported by the Australian Research Council (ARC) (under Grant DP140103171) and National Natural Science Foundation of China (NSFC) (under Grant 61472263).

7. REFERENCES

[1] J. Teevan, D. Ramage, and M. R. Morris. #twittersearch: A comparison of microblog search and web search. In *WSDM*, pages 35–44, 2011.

[2] W. Shen, J. Wang, P. Luo, and M. Wang. Linking named entities in tweets with knowledge base via user interest modeling. In *KDD*, pages 68–76, 2013.

[3] R. Mihalcea and A. Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *CIKM*, pages 233–242, 2007.

[4] D. Milne and I. H. Witten. Learning to link with wikipedia. In *CIKM*, pages 509–518, 2008.

[5] X. Han and J. Zhao. Named entity disambiguation by leveraging wikipedia semantic knowledge. In *CIKM*, pages 215–224, 2009.

[6] X. Han and J. Zhao. Structural semantic relatedness: A knowledge-based method to named entity disambiguation. In *ACL*, pages 50–59, 2010.

[7] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of wikipedia entities in web text. In *KDD*, pages 457–466, 2009.

[8] X. Han, L. Sun, and J. Zhao. Collective entity linking in web text: A graph-based method. In *SIGIR*, pages 765–774, 2011.

[9] W. Shen, J. Wang, P. Luo, and M. Wang. Linden: Linking named entities with knowledge base via semantic knowledge. In *WWW*, pages 449–458, 2012.

[10] E. Meij, W. Weerkamp, and M. de Rijke. Adding semantics to microblog posts. In *WSDM*, pages 563–572, 2012.

[11] A. Davis, A. Veloso, A. S. da Silva, W. Meira Jr., and A. H. F. Laender. Named entity disambiguation in streaming data. In *ACL*, pages 815–824, 2012.

[12] X. Liu, Y. Li, H. Wu, M. Zhou, F. Wei, and Y. Lu. Entity linking for tweets. In *ACL*, pages 1304–1311, 2013.

[13] Y. Li, C. Wang, F. Han, J. Han, D. Roth, and X. Yan. Mining evidences for named entity disambiguation. In *KDD*, pages 1070–1078, 2013.

[14] P. Ferragina and U. Scaiella. Tagme: On-the-fly annotation of short text fragments (by wikipedia entities). In *CIKM*, pages 1625–1628, 2010.

[15] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: Understanding microblogging usage and communities. In *WebKDD/SNA-KDD*, pages 56–65, 2007.

[16] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[17] L. Chen, A. Gupta, and M. E. Kurul. Stack-based algorithms for pattern matching on dags. In *VLDB*, pages 493–504, 2005.

[18] S. Tribl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD*, pages 845–856, 2007.

[19] H. Yildirim, V. Chaoji, and M. J. Zaki. Grail: Scalable reachability index for large graphs. *PVLDB*, 3(1-2):276–284, 2010.
[20] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *SIGMOD*, pages 253–262, 1989.
[21] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu. Dual labeling: Answering graph reachability queries in constant time. In *ICDE*, pages 75–75, 2006.
[22] Y. Chen and Y. Chen. An efficient algorithm for answering graph reachability queries. In *ICDE*, pages 893–902, 2008.
[23] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficiently answering reachability queries on very large directed graphs. In *SIGMOD*, pages 595–608, 2008.
[24] S. J. van Schaik and O. de Moor. A memory efficient reachability data structure through bit vector compression. In *SIGMOD*, pages 913–924, 2011.
[25] Y. Chen and Y. Chen. Decomposing dags into spanning trees: A new way to compress transitive closures. In *ICDE*, pages 1007–1018, 2011.
[26] S. Seufert, A. Anand, S. Bedathur, and G. Weikum. Ferrari: Flexible and efficient reachability range assignment for graph indexing. In *ICDE*, pages 1009–1020, 2013.
[27] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast computing reachability labelings for large graphs with high compression rate. In *EDBT*, pages 193–204, 2008.
[28] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3hop: A high-compression indexing scheme for reachability query. In *SIGMOD*, pages 813–826, 2009.
[29] J. Cai and C. K. Poon. Path-hop: Efficiently indexing large graphs for reachability queries. In *CIKM*, pages 119–128, 2010.
[30] J. Cheng, Z. Shang, H. Cheng, H. Wang, and J. X. Yu. K-reach: Who is in your small world. *PVLDB*, 5(11):1292–1303, 2012.
[31] J. Cheng, S. Huang, H. Wu, and A. W. Fu. Tf-label: A topological-folding labeling scheme for reachability querying in a large graph. In *SIGMOD*, pages 193–204, 2013.
[32] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, pages 1601–1606, 2013.
[33] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.
[34] R. Jin and G. Wang. Simple, fast, and scalable reachability oracle. *PVLDB*, 6(14):1978–1989, 2013.
[35] I. Witten and D. Milne. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *AAAI*, pages 25–30, 2008.
[36] G. Li, J. Hu, J. Feng, and K. Tan. Effective location identification from microblogs. In *ICDE*, pages 880–891, 2014.
[37] C. Li, J. Weng, Q. He, Y. Yao, A. Datta, A. Sun, and B. Lee. Twiner: Named entity recognition in targeted twitter stream. In *SIGIR*, pages 721–730, 2012.
[38] D. M. de Oliveira, A. H. F. Laender, A. Veloso, and A. S. da Silva. Fs-ner: A lightweight filter-stream approach to named entity recognition on twitter data. In *WWW*, pages 597–604, 2013.
[39] K. Takeuchi and N. Collier. Use of support vector machines in extended named entity recognition. In *CONLL*, pages 1–7, 2002.
[40] H. Isozaki and H. Kazawa. Efficient support vector classifiers for named entity recognition. In *COLING*, pages 1–7, 2002.
[41] H. L. Chieu and H. T. Ng. Named entity recognition: A maximum entropy approach using global information. In *COLING*, pages 1–7, 2002.
[42] O. Bender, F. J. Och, and H. Ney. Maximum entropy models for named entity recognition. In *CONLL*, pages 148–151, 2003.
[43] J. R. Curran and S. Clark. Language independent ner using a maximum entropy tagger. In *CONLL*, pages 164–167, 2003.
[44] G. Zhou and J. Su. Named entity recognition using an hmm-based chunk tagger. In *ACL*, pages 473–480, 2002.
[45] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *CONLL*, pages 188–191, 2003.
[46] A. Ritter, S. Clark, Mausam, and O. Etzioni. Named entity recognition in tweets: An experimental study. In *EMNLP*, pages 1524–1534, 2011.
[47] X. Liu, S. Zhang, F. Wei, and M. Zhou. Recognizing named entities in tweets. In *HLT*, pages 359–367, 2011.
[48] X. Liu, M. Zhou, F. Wei, Z. Fu, and X. Zhou. Joint inference of named entity recognition and normalization for tweets. In *ACL*, pages 526–535, 2012.

APPENDIX

A. ADDITIONAL RELATED WORK

Named Entity Recognition (NER). One of the prerequisites to entity linking is Named Entity Recognition, namely detecting named entities from texts written in natural languages. Existing approaches to NER can be classified into two categories: linguistic-based and knowledge-based.

Linguistic-based approaches are the current dominant techniques for addressing the NER problem. Most of them formulate NER as a *sequential labeling* task, and apply various machine learning algorithms, including Support Vector Machine (SVM) [39, 40], Maximum Entropy Model (ME) [41, 42, 43], Hidden Markov Model (HMM) [44], and Conditional Random Field (CRF) [45], to determine the NE labels of tokens collectively. The features they employ are basically linguistic-based such as capitalization, digitalization, punctuation, part-of-speech tags, and so forth. Despite the already satisfying performance that linguistic-based approaches to NER have achieved on news articles and Web documents, they cannot be applied directly to short texts like tweets. This is partially due to the fact that tweets are informal and error-prone, making traditional linguistic features (e.g., capitalization) inapplicable. Recently, much work [46, 47, 48] has been devoted to designing linguistic features to capture tweets' unique characteristics and training tweet-specific models to extract named entities. For example, Ritter et. al [46] explore features including retweets, @usernames, hashtags, URLs, and Brown clustering results to train tweet-specific POS tagger (T-POS) and shallow parsing (T-CHUNK), whose results are then regarded as features to train a CRF model for named entity recognition (T-NER). Liu et. al [47] leverage cross-tweet information by using a KNN-based classifier to conduct word-level pre-labeling, and then feed those information into a CRF model to conduct finer-grained NER. However, due to their supervised nature, these approaches require a vast amount of labeled data which is usually expensive to come by.

Knowledge-based approaches [37, 38] to NER are unsupervised, making them more feasible for handling texts in a streaming manner. A knowledgebase (also called gazetteer, lexicon, or dictionary) is required in such methods, and named entity recognition is conducted by simply checking for existence or frequency of a phrase in the knowledgebase. For example, the widely-used Longest-Cover method searches for longest terms contained in the knowledgebase while scanning a text. Like many recent attempts to entity linking in short texts [14, 2], we also adopt the knowledge-based approach as a pre-step to extract entity mentions, considering its simplicity and real-time nature. Note that most knowledge-based approaches implicitly require candidate phrases to exactly match at least one element in the knowledgebase, but some flexibility can be added by allowing for word stemming or fuzzy matching.

B. METHODOLOGY

In this section, we prove the theorems proposed in this work to verify the correctness of our algorithms.

B.1 Proof of Theorem 1.

Theorem 1 claims that there exists a *len*-hop shortest path from u to v which passes through u' followee t only if the shortest path distance from t to v is $len-1$ hops. We restate this theorem formally as below:

THEOREM 3. $\forall t \in F_u, d_{uv} = len \wedge t \in u \rightsquigarrow^* v \Rightarrow d_{tv} = len - 1$, *or simply* $\forall t \in F_u, t \in u \rightsquigarrow^* v \Rightarrow d_{tv} = d_{uv} - 1$.

where F_u denotes the collection of u's followees, and d_{uv} the shortest path distance from u to v. $t \in u \rightsquigarrow^* v$ means that u's followee t participates in a *len*-hop path from u to v.

PROOF. Assume by contradiction that $d_{tv} \neq len - 1$, which means either t cannot reach v or the shortest path distance from v to t is longer or shorter than $len - 1$ hops. 1) If v is not reachable from t (i.e., $t \not\rightsquigarrow v$), then u cannot reach v through its followee t (i.e., $t \notin u \rightsquigarrow v$); 2) If the shortest path distance from t to v is longer than $len - 1$ hops (i.e., $d_{tv} > len - 1$), then the distance of path $u \rightarrow t \rightsquigarrow v$ is larger than *len*, implying that t cannot participate in a *len*-hop shortest path from u to v (i.e., $t \notin u \rightsquigarrow^* v$); 3) If the shortest path distance from t to v is shorter than $len - 1$ hops (i.e., $d_{tv} < len - 1$), then the distance of path $u \rightarrow t \rightsquigarrow v$ is smaller than *len*, which means a shorter path from u to v has been detected (i.e., $d_{uv} < len$). □

COROLLARY 1. $\forall t, d_{uv} = Len \wedge d_{ut} = len \wedge t \in u \rightsquigarrow^* v \Rightarrow d_{tv} = Len - len$, *or simply* $\forall t, t \in u \rightsquigarrow^* v \Rightarrow d_{tv} = d_{uv} - d_{ut}$.

where d_{uv} is the shortest path distance from u to v, and $t \in u \rightsquigarrow^* v$ means that t participates in the shortest path from u to v. Corollary 1 can be proved similarly as in the above theorem.

B.2 Proof of Theorem 2.

Theorem 2 claims that if the shortest path from s to v passes through s's followee f, and v lies in the shortest path from s to t, then f participates in the shortest path from s to t. We restate this theorem formally as below:

THEOREM 4. $\forall f \in F_s, f \in s \rightsquigarrow^* v \wedge v \in s \rightsquigarrow^* t \Rightarrow f \in s \rightsquigarrow^* t$

where F_s denotes the collection of s's followees, and $f \in s \rightsquigarrow^* t$ means that f participates in the shortest path from s to t.

PROOF. Assume the shortest path distance from s to t is $d_{st} = Len$. Since s subscribes to f, it suffices to prove that there exists a $(Len - 1)$-hop path from f to t. From $f \in s \rightsquigarrow^* v$ and $v \in s \rightsquigarrow^* t$, we can obtain that f can reach v which in turn reaches t. In other words, we have successfully detected a path from f to t, namely $f \rightsquigarrow v \rightsquigarrow t$. Now we only need to show that the distance of path $f \rightsquigarrow v \rightsquigarrow t$ is $Len-1$ hops. Assume the shortest path distance from s to v is $d_{sv} = len$. According to Corollary 1, $d_{fv} = d_{sv} - 1 = len - 1$ and $d_{vt} = d_{st} - d_{sv} = Len - len$. Hence, there exists a path $f \rightsquigarrow v \rightsquigarrow t$ whose distance is $d_{fv} + d_{vt} = Len - 1$. This completes our proof. □

C. ADDITIONAL EXPERIMENTS

C.1 Generalizability of our system

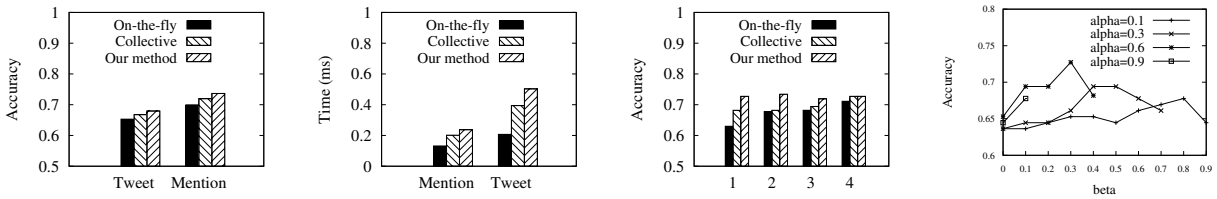
Different microblogging sites. We used Twitter dataset in our experiments considering the wide-spread adoption and the availability of large-scale tweet dataset for empirical study in this community. However, our approaches can apply to solve entity linking problem on any microblogging social network (e.g., Facebook, Google+, Pinterest, Chinese Sina Weibo, etc), since we do not exploit any features that are unique to Twitter.

To verify this, we collected 23.3 million tweets published between January 2012 and October 2012 as well as the followee-follower network using Chinese Sina Weibo's API. We downloaded the July 2014 version of Chinese Wikipedia from which 5.1 million mentions and 3.5 million entities were extracted to construct our knowledgebase. We conducted a collective entity linking [2] on part of the Sina Weibo dataset published by active users to complement the original knowledgebase, and used postings of inactive users for testing. The experimental results are consistent with those found in Twitter. Due to limitation of paper length, we only present the comparison of our framework and existing state-of-the-art approaches to entity linking in terms of both accuracy and efficiency, as illustrated in Fig. 6 (a) and (b) respectively. From Fig. 6 (a) we can see that our approaches still outperform existing On-the-fly method [14] and Collective method [2] when applied to Chinese Sina Weibo, although the improvement is slightly smaller than that achieved in Twitter (3.7% mention accuracy improvement and 2.6% tweet accuracy improvement in Sina Weibo vs. 6% mention accuracy improvement and 4.5% tweet accuracy improvement in Twitter - Fig. 4 (a)). Postings in Sina Weibo usually contain more entities (2.3 entities per tweet in our Sina Weibo dataset) than Twitter, making it much more reliable to consider topical coherence for entity disambiguation, which in turn leads to better performance of the On-the-fly method in Sina Weibo compared with Twitter. Fig. 6 (b) demonstrates that our framework can link entities of a tweet in Sina Weibo within 0.5 milliseconds in average. This is also sufficiently fast for most real-world applications. For example, Sina Weibo reported that about 100 million messages were posted each day in the year 2012[6], which requires an entity linking framework to handle each tweet within 2 milliseconds. In this sense, our approaches are also feasible for real-time entity linking in Chinese Sina Weibo.

Varying tweet length. We do not theoretically restrict the number of entities per posting in our framework, though we believe there should not be too many entity mentions in each tweet in a microblogging site like Twitter. We conduct an experiment in the paper to study the differences between our framework and existing state-of-the-art approaches with varying tweet lengths. We divide our test dataset into 4 parts according to tweet length (i.e., number of entity mentions per tweet) and then apply these methods to each part. As illustrated in Fig. 6 (c), the accuracy of our framework for entity linking remains quite stable when tweet length ranges from 1 to 4. This is mainly due to the fact that we handle each entity mention separately without considering their interdependence. We notice that the superiority of our method over other content-based counterparts is even more considerable when the target tweet contains only 1 entity mention. In this case, topical coherence adopted in [14] is no longer applicable, making the entity linking results unsatisfactory. [2] can improve the situation a little bit by estimating users' interest from their historical postings and linking all the entities mentioned by a single user collectively. However, such an improvement is insignificant for information seekers with few tweeting histories. Although Fig. 6 (c) shows an accuracy improvement of [14] and [2] when tweet length increases, our framework is still in a very competitive position since most microblogging sites (e.g., Twitter, Chinese Sina Weibo, etc.) impose a words limitation on their content, which in turn limit the number of entities per posting.

Different entity categories. It is also important to see how our framework performs on different categories of entities, in order to verify its generalizability. We asked annotators to manually classify the entities mentioned in our test dataset into 5 categories - Person

[6] https://en.wikipedia.org/wiki/Sina_Weibo



(a) effectiveness of our framework compared with state-of-the-art methods on Sina Weibo.

(b) efficiency of our framework compared with state-of-the-art methods on Sina Weibo.

(c) effectiveness of our framework when tweet length varies.

(d) sensitivity of our framework to parameters.

Figure 6: Additional experimental results.

(71.35%), Location (8.38%), Company (2.6%), Product (2.27%), and Movie&Music (15.4%), and evaluate the accuracy of entity linking respectively. We find Location category gets the best accuracy (74.32%) and Movie&Music category has the worst (71.32%), which demonstrates that the performances of our framework on these five categories are similar and also irrelevant to the number of entities in each category. This is an expected result, since we do not employ any features specific to a particular category of entities.

C.2 Sensitivity of our system to parameters

In this work, we combine user interest, entity recency, and entity popularity to derive an overall ranking of candidate entities for a target entity mention. We adopt a linear combination of these three features in the current model, and use α, β and γ to reflect the relative contributions of these features to the final scoring of candidate entities. From Table 4 we can see that the performance of user interest estimation using social interactions is better than that obtained using traditional content-based approaches, and that entity recency performs better than entity popularity, since recency is a time-dependent feature which measures an entity's recent popularity. By combining all these features, our framework achieves the highest accuracy. In order to get a better insight of the sensitivity of our framework to the values of α, β and γ, we conduct an experiment by setting α to 0.1, 0.3, 0.6, and 0.9 respectively and measuring the performance of our system with various combinations of β and γ. Fig. 6 (d) shows that our method is indeed sensitive to the settings of these parameters. For each value of α, the best performance can be achieved only when neither β nor γ are equal to 0, verifying the importance of both entity recency and entity popularity for entity linking. Furthermore, the peak value of accuracy is always obtained on the right side of Fig. 6 (d) (i.e., when β is larger than γ), which indicates the superior of entity recency over entity popularity. This is consistent with the result in Table 4. In practice, these three parameters can be manually defined or automatically learned using existing machine learning methods. We choose to set them manually due to lack of large-scale training dataset. Section 5.1 reports the best combination of α, β and γ according to the experimental results.

D. DISCUSSION

It is important to think about the impact of vocabulary on linking quality, considering the dynamic feature of microblogging sites. Usually the knowledgebase should periodically check if new entities or new meanings for old entities appear in the Web and update its entity database accordingly. If new entities/meanings do appear but have not been reflected in knowledgebase, these entity mentions cannot be detected from tweets or be linked to the new meanings, since the nature of entity linking problem is to recognize and link an entity mention to the most appropriate entity meaning that already existed in the knowledgebase. The knowledgebase can be updated manually (e.g., Wikipedia) or automatically (e.g., Probase). There have been a lot of interesting discussions on the update policy for knowledgebase, which is out of the scope of this work. However, there are indeed two issues we need to consider, as detailed below.

First, we should avoid the false-positive error before the knowledgebase is updated. More specifically, an entity mention should not be linked to any of the existing entity meanings if the target user is found to have no interest in existing meanings of the entity. Since our framework adopts a linear combination of user interest, entity recency and entity popularity, any candidate entity meaning the target user is not interested in will receive an overall score no greater than $\beta + \gamma$. Therefore, we can set a threshold at $\beta + \gamma$ and only output top-k entities whose score exceeds the threshold. In this way, we avoid the false-positive error even before knowledgebase has been updated properly. Moreover, when the entity linking result is empty, it is most probable that the target entity mention should be linked to a new meaning which does not exist in the current knowledgebase. In this case, we can interactively ask the user to add a new entity meaning. By this means our framework can also help updating the knowledgebase in proactive manner.

Second, we should avoid the true-negative error after the knowledgebase is updated. This means the new entity meanings should be able to be linked to after knowledgebase updates. Since we complement the original knowledgebase using historical broadcastings, the newly-generated tweets containing mentions to new entity meanings have not been complemented to the knowledgebase. Therefore, a system warm-up is required, which can also been done through the aforementioned interactive process. In particular, when no entity linking result can be derived or the target user finds the top-k entity meanings unsatisfactory, we provide her with the new meaning and allow her to decide whether it is the desired one.