LBSNRank: Personalized PageRank on Location-based Social Networks *

Zhaoyan Jin
National University of Defense Technology
Changsha, China
jinzhaoyan@163.com

Dianxi Shi
National University of Defense Technology
Changsha, China
dxshi@nudt.edu.cn

Quanyuan Wu
National University of Defense Technology
Changsha, China
wqy.nudt@gmail.com

Huining Yan
National University of Defense Technology
Changsha, China
yhnbj@qq.com

Hua Fan
National University of Defense Technology
Changsha, China
fh.nudt@gmail.com

ABSTRACT
Different from traditional social networks, the location-based social networks allow people to share their locations according to location-tagged user-generated contents, such as checkins, trajectories, text, photos, etc. In location-based social networks, which are based on users' checkins, people could share his or her location according to checkin while visiting around. However, people's locations change frequently and the rankings of people change dynamically too, which makes ranking on graphs a challenging work. To address this challenge, we propose the LBSNRank algorithm on graphs with nodes whose contents change dynamically. To validate our algorithm on real datasets, we have crawled and analyzed a dataset from the Dianping website. Experiments on this real dataset show that our LBSNRank algorithm performs better than traditional personalized PageRank in efficiency.

Author Keywords
location-based social networks, personalized PageRank, Monte Carlo method, random walk, MapReduce.

ACM Classification Keywords
H.2.8 Database Applications: Spatial databases and GIS.

General Terms
Algorithms, Experimentation, Performance

INTRODUCTION

With the rapid development of mobile devices and wireless broadband access, the market of mobile internet grows up quickly and lots of new services come up. As two of the most important mobile internet services, location-based services and mobile social network services developed separately in the past. But nowadays, we have been seeing a convergence of the two, and a new kind of social networks, called location-based social network (LBSN for short), is becoming increasingly popular and hundreds of millions of users are active on a daily basis. In traditional social networks, users maintain their relationships mainly among friends in the virtual world, while in LBSN, users can also know new friends online and enhance their relationships according to the offline activities. Obviously, the LBSN offers us a better experience of communication. For more about LBSN, one can refer to Zheng [28] for details.

The LBSN services not only allow users to tweet, reply or retweet, but also allow users to post their locations while tweeting. For example, Foursquare[1] and Dianping[2] are two popular web sites of this kind, which allow users to checkin while tweeting. As time goes on, each user has a history of locations which reflects his or her interests, hobbies, habits, etc. A simple LBSN can be seen in figure 1. Scellato et al. [22, 23] find that, most geographical distances among friends are short-distance links and users like to make friends with nearby people in a specific location. So how to choose popular locations and how to ascertain influential people in some specific location are two important topics in LBSN. Consider the following two scenarios.

(1) As a traveler, you are prepared to visit the Great Wall in Beijing. You can find some influential people in Beijing especially in the area of the Great Wall in advance, and then you can browse their comments, talk with them, and even make friends and live with them in the physical world.
(2) After visiting the Great Wall, you also want to visit some other popular places in Beijing.

*This work was supported in part by the National Significant Science and Technology Special Project of China (Nos. 2011ZX03002-004-01 and 2009ZX01043-002-004) and the National Natural Science Foundation of China (No. 90818028.)

UbiComp '12, Sep 5-Sep 8, 2012, Pittsburgh, USA.

[1] www.foursquare.com
[2] www.dianping.com

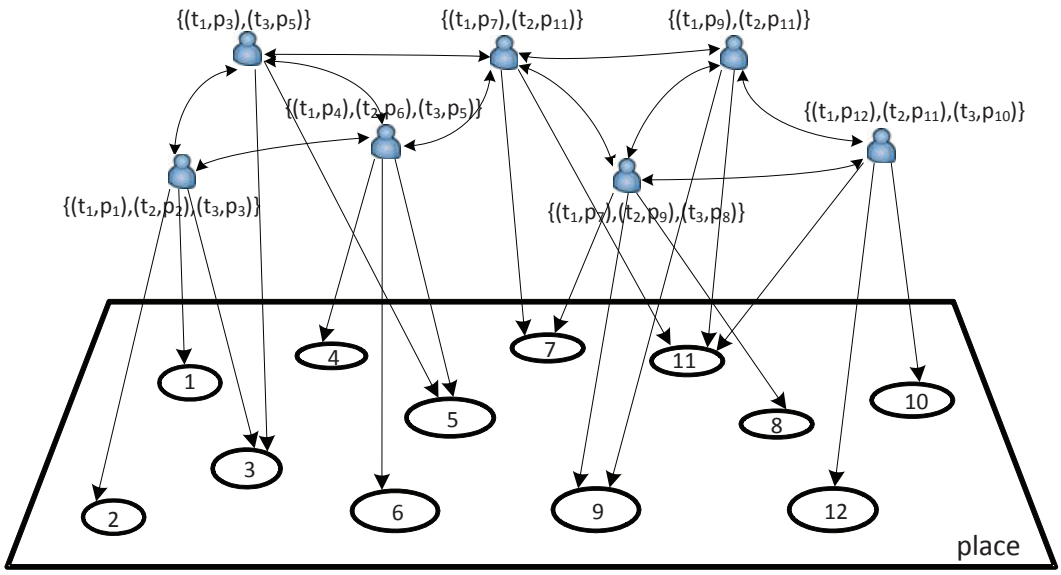


Figure 1. A simple LBSN

In order to find popular places and influential people in specific location, ranking algorithms, such as PageRank [17], HITS [14], Monte Carlo method [16, 10] and the algorithms derived from them, can be used to compute the rankings of people in a LBSN. These algorithms deal with graphs with static nodes, which assume that the contents of the nodes in a graph never change. However, in LBSN, people's locations change frequently and the rankings of people and locations change dynamically too, which makes ranking on such graphs a challenging work.

In this paper, we aim to rank users and locations with respect to some specific location in a LBSN, which means ranking on a graph based on users' relationships and location histories. As there are many users and locations in a LBSN, ObjectRank [4] suggests precomputing all rankings offline in order to answer users' queries quickly. In order to save computing and storage resources, HubRank [6] suggests precomputing some fixed important rankings offline. However, in a LBSN, the checkin records of users change with time frequently, and it isn't advisable to preprocess a fixed location set, and thus we adapt the offline location set dynamically according to locations' popularity. We propose the LBSNRank algorithm for graphs with nodes whose contents change dynamically, have crawled and analyzed a real dataset from the Dianping website, and validate the efficiency of the proposed LBSNRank algorithm.

The rest of the paper is organized as follows. In section 2, we introduce some background for the PageRank and the Monte Carlo method based PageRank and review some related work. Our LBSNRank algorithm is given in section 3. In section 4, we analyze some characteristics of our dataset, and experiments are given in section 5. Conclusion and future work is given in section 6.

BACKGROUND AND RELATED WORK

In this section, we first introduce some background for the PageRank algorithm and the Monte Carlo method of PageRank computing, and then review some work related to ranking on graphs.

Background

We assume to have a weighted graph $G = (V, E)$ with n nodes and m edges, and the weight on an edge (u, v) ($(u, v) \in E$) is denoted with $w_{u,v}$. For the sake of simplifying the presentation of the formulae, we assume that the weights on the outgoing edges of each node sum up to 1.

PageRank

The PageRank method computes the stationary distribution of a random walk. Starting from a source node, we walk randomly in a graph. At each step, we jump to a personalized node with the probability ϵ, and walk along the current node's outgoing edges with the probability $1 - \epsilon$. When the number of steps is big enough, no matter which source node we choose, the stationary probability of each node tends to be a constant, and is called its PageRank score.

With respect to a source node u, the PageRank score $\pi_u(v)$ of node v satisfies:

$$\pi_u(v) = \epsilon\delta_u(v) + (1 - \epsilon) \sum_{\{w|(w,v)\in E\}} \pi_u(w)w_{w,v} \quad (1)$$

At each step, we jump to a personalized node with the probability ϵ, so how to choose the personalized node decides the classification of PageRank. The naive PageRank is that we choose the personalized node fully randomly, which mean that $\delta_u(v) = \frac{1}{n}$ for all $v \in V$, the topic-sensitive PageRank is that we choose different nodes for different probabilities, which is $\delta_u(v) = p_v$ and $\sum p_v = 1$ for all $v \in V$, and the

personalized PageRank is the same as PageRank except that we jump to the source node at all jumps, where $\delta_u(v) = 1$ if $u = v$, and 0 otherwise.

Monte Carlo Method of PageRank Computing

In order to compute the personalized PageRank Score of each node, we can either use linear algebraic techniques, such as Power Iteration [17], or approximate the personalized PageRank score using the Monte Carlo method, which simulates several real random walks and then estimates the stationary distribution with the empirical distribution of the performed random walks. Based on this idea, Litvak [16] and Fogaras et al. [9, 10] propose the following approximation method for personalized PageRank: Starting from each node $u \in V$, do a number, R, of random walks and at each random step, stop with the probability of ϵ, and do the random walk with a probability of $1 - \epsilon$. When it stops, The nodes visited are called "fingerprints", and the length of a fingerprint conforms to the geometrical distribution Geom(ϵ). Then, the frequency of visits to each node in all fingerprints could approximate the PageRank score.

Related Work

Webpage Ranking: Webpage ranking is a hot-spot research in the World Wide Web. Classical methods, such as HITS [14] and PageRank [17] are both link analysis algorithms. HITS assumes that the ranking of a page contains authority and hub, where authority means that a site receives many citations and the citation from important sites weights more than less important sites, and hub means that a site links to many authoritative sites. Because the computation process is carried out online and needs lots of computation resources, it can not satisfy users' queries timely. In contrast, PageRank computes only one measure of ranking for each page offline. The heuristic underlying this is that the ranking of a page is proportional to its parents' ranking, and inversely proportional to the number of its parents' outgoing edges. The PageRank method has a better efficiency, which is the reason why google successes. Following the success of PageRank, a lot of link analysis algorithms, such as Block-level PageRank [5], HostRank[6], hierarchical rank [26], etc., are proposed. However, there is a common disadvantage in link analysis algorithms, that is, the ranking of a page depends on the link structure of webpages and they ignore the contents of the query at hand.

Personalized PageRank: Chakrabarti et al. [7], Pennock et al. [18] and Richardson et al. [19] demonstrate that the properties of web graphs are sensitive to page topics, and Haveliwala et al. [12, 11] propose the TS-PageRank (Topic-Sensitive PageRank) and the personalized PageRank. Instead of using a single PageRank score to represent the importance of a page, TS-PageRank calculates a vector of PageRank scores for every page according to the 16 topics in ODP[3]. The TS-PageRank contains two processing steps, offline and online. In the offline step, the algorithm computes a PageRank score for each topic based on different transition matrix, while in the online step, it computes the linear combination of the vector for every page according to the query context and returns the ordered pages. The TS-PageRank method deals with queries that are relevant to a limited number of topics. For fully personalized PageRank, HubRank [6] and BinRank [13] propose approaches that generate a vector for every possible query term. Since the query terms are so great that they can not be preprocessed completely, so they classify the query terms, such as fixed classification and classification based on similarity, and preprocess the classified ones.

Monte Carlo Method: The Monte Carlo method proposed by Litvak [16] and Fogaras et al. [9, 10] is an efficient method to compute personalized PageRank. It simulates several real random walks with fingerprints and then approximates the personalized PageRank with the empirical distribution of the performed walks. However, some of the fingerprints maybe very long, and hence limit the efficiency of parallel algorithms. Sarma et al. [21] propose the idea of doing random walks of fixed length starting from each node, which improves the parallel efficiency greatly. The Monte Carlo method is not only efficient, but also allows to perform continuous update of the PageRank as the structure of the graph changes [3]. Avrachenkov et al. [1] demonstrate that the Monte Carlo method provides good estimation of PageRank for relatively important pages already after one iteration. As the MapReduce programming model becomes more and more popular, the optimal implementation of Monte Carlo approximation of personalized PageRank vectors of all the nodes in a graph is studied by Bahmani et al. [2].

Social Entity Ranking: Besides webpage ranking, social entity ranking in social networks also considers ranking on large-scale social graphs. TunkRank [24] and TwitterRank [25], both of which are variants of PageRank, measure the rankings of users based on a graph constructed by the "following" relationships in Twitter. The difference is that TwitterRank considers both the topic similarity between users and the following relationships, whereas TunkRank measures rankings on Twitter based on how much attention one's followers can actually give him. In addition, the IP-influence algorithm [20], similar to HITS, takes influence and passivity into consideration while it calculates rankings in social graphs. In social networks, users can tweet, retweet and reply about a topic. These actions can also construct graphs, such as the user-tweet graph [15] and the action-based user influence graph [27]. Based on these graphs, we can also rank users according to their corresponding definitions.

Travel Recommendation in LBSN: Location histories of people in LBSN ont only reflect where they have gone, but also imply the location correlation in people's daily lives [29]. With this correlation, a lot of valuable services, such as travel recommendation, sales promotion and bus route planning, could be enabled. Zheng et al. [30, 31] study the problem of locations and travel sequences recommendation from user-generated GPS trajectories. They extract stay points from these trajectories, construct a Tree-Based Hierarchical Graph, and then mine points of interest (POIs) and classic travel sequences. In order to compute the score of

[3]http://www.opendirectoryproject.com/

each POI, they construct a HITS-based inference model. In that model, locations and users have a mutual reinforcement relationship. But in this paper, we rank users according to their relationships and sort locations according to their popularity. As the links of graph and contents of nodes change with time, we adjust the offline location set dynamically in order to shorten the response time to users' queries.

LBSNRANK

In this section, we introduce some notations, state the problem of ranking on LBSN, and describe the proposed LBSNRank algorithm.

Notations and Problem Definition

In a LBSN, we view users as nodes, and the following / followed relations among users as directed edges, thus we can get a weighted directed graph $G = (V, E)$, where $|V| = n$ and $|edge| = m$. We denote the weight on an edge (u, v) ($(u, v) \in E$) with $w_{u,v}$ ($w_{u,v} = \frac{1}{out(u)}$, where $out(u)$ means the number of out-links of u). We also have a location history L_G, where each node $u \in V$ has a list l_u ($l_u \in L_G$) of location-timestamp pair (t, p), i.e., $l_u = \{(t_i, p_i)\}$, which means that the user u visited the location p_i at time t_i.

We begin by defining the **ranking of a user** with respect to some location in some period:

Definition 1. Given a social graph G and its corresponding location history L_G, the ranking score of a node u, in location p between t_1 and t_2 with the place-timestamp pairs $l_{u,p}(t_1, t_2) = \{(t_i, p)|(t_1 \leq t_i \leq t_2)\}$, is decided by its personalized PageRank:

$$\pi_{p,t_1,t_2}(v) = \epsilon\delta_{p,t_1,t_2}(v) + (1-\epsilon) \sum_{\{o|(o,v)\in E\}} \pi_{p,t_1,t_2}(o) w_{o,v} \tag{2}$$

Where $\delta_{p,t_1,t_2}(v) = \frac{|l_{v,p}(t_1,t_2)|}{\sum_{o\in V} |l_{o,p}(t_1,t_2)|}$.

Next, we define the **ranking of a location** in some period:

Definition 2. The ranking of a location p in the period between t_1 and t_2 is proportional to the number of visitors, the ranking and visited times of each visitor, that is:

$$r_p(t_1, t_2) = \sum_{u\in V} \pi_{p,t_1,t_2}(u) \times |l_{u,p}(t_1, t_2)| \tag{3}$$

LBSNRank Algorithm

In this section, we describe our LBSNRank algorithm for fully personalized PageRank. In LBSN, there are such many locations that people have visited that it is impossible to preprocess rankings with respect to all locations. As a matter of fact, most people have visited only a few of locations, so we need only to preprocess rankings of people for those popular locations offline, and compute rankings of people for less popular locations online. Because the popularity of locations changes frequently, the location set that we preprocess offline changes frequently, too. In the following algorithm, we adjust the location set that will be preprocessed offline at every iteration.

The LBSNRank algorithm:
1. determine the location set Sp that will be preprocessed;
2. compute the ranking of the location $p \in Sp$;
3. compute the ranking of people for each location $p \in Sp$, and go to step 1 for next iteration;

Determination of Location Set

There are two reasons that we can't compute all personalized PageRank scores offline. One is that the set of locations is so big that computing and storing all personalized PageRank scores would need lots of computing and storage resources, and even a small dataset would take several days in our experiments. The other reason is that people update their locations so frequently that if we don't compute the required personalized PageRank timely, then the results returned would be meaningless. In order to answer queries timely, we choose a set of popular locations (details is in Section), for each of which we compute its personalized PageRank offline. This not only answers users' queries timely with preprossed results but also reduces the computing and storage resources by an order of magnitude or more.

Ranking for Locations

As time goes on, the popularity or ranking for each location changes with time. For example, an important sport activity may make some location popular at that time, whereas the change of seasons may make other locations comfortable or attractive in fixed seasons. Initially, we assume all rankings of people are equal, that is $\pi_p(u) = \frac{1}{n}$, for all $p \in Sp$ and $u \in V$, and the ranking of location p is $r_p = \frac{1}{n}\sum_{u\in V} |l_{u,p}|$. Since the second iteration, the rankings of locations are computed according to Definition 2.

Ranking for Users

To compute the rankings of people with respect to some location, we employ the fixed length random walks ([21]) on the reduced subgraph. To reduce the graph in some period about a given location, we select all users who visit that location in that period, and expand their neighbours, including inlinks and outlinks. Algorithm 1 describes the details of our random walks. In algorithm 1, the personalized vector $V_p = \delta_{p,t_1,t_2}$, which can be calculated according to equation 2, and the choosing of the timestamp pairs t_1 and t_2 between two consecutive iterations depends on your actual requirements and the ability of your platform.

Query Processing

For a given query, if it requires some locations, the LBSNRank algorithm returns high ranking locations from the location set, and if it requires some people in a location, then the LBSNRank algorithm returns top-k people relevant to that location. As the LBSNRank algorithm computes rankings with respect to only a few of locations, the query may require both preprocessed and unprocessed locations. If all the required locations is preprocessed, the LBSNRank algorithm returns a linear composition of them. Otherwise,

Algorithm 1 MonteCarloK(G_r, Vp, ϵ, k)

Input: A reduced subgraph $G_r = (V, E)$, the personalized vector V_p, the possibility ϵ for restart and the length k for each fingerprint;
Output: A list l of $(v, frequency)$, for all $v \in V$ in G_r;

```
let l = {(v,0)|∀v ∈ V};
for all v ∈ V do
    let current = v;
    for i = 1 to k do
        if Random.nextDouble() < ε then
            //next random walk;
            next = current.neighbours.Random();
            (next, frequency) = (next, frequency + 1);
            current = next;
        else
            //restart;
            for i = 1 to |Vp| do
                if Random.nextDouble() <= Vp_i then
                    next = i;
                    (next, frequency) =
                    (next, frequency + 1);
                    current = next;
                end if
            end for
        end if
    end for
end for;
```

if it requires unprocessed locations, then the LBSNRank algorithm computes unprocessed locations online, and returns a linear composition of them. By default, queries are accompanied by the latest timestamp pairs, but if a query requires older ranking, the LBSNRank algorithm deals with it online.

DATASET

To evaluate our method in a real LBSN, we have crawled the Dianping website to collect a dataset of users, their social links and checkin histories. We have crawled about 200,000 users from the Dianping website. The dataset are crawled from Dec. 19, 2011 to Feb. 26, 2012, and table 1 lists some basic statistics of this dataset.

Table 1. Statistics of the Dianping Dataset

# users	# links	# checkins
204,074	926,720	2,730,072
# cities	# districts	# POIs
347	1,691	313,565

Time-Checkin Distribution

The Dianping website prevents us from crawling the whole checkin records, so we can't have a complete view of users' checkin histories. In our dataset, the checkins are from Jan. 2011 to Feb. 2012, and the distribution of which is in figure 2. As can be seen from the figure that, the number of checkins grows up quickly from Jan. 2011 to Aug, 2011, and after that, it grows smoothly. The reason is that the Dianping website has been developing quickly since its introduction of location-based services. Moreover, the crawling is begin with the homepage, so most of the dataset are up-to-date.

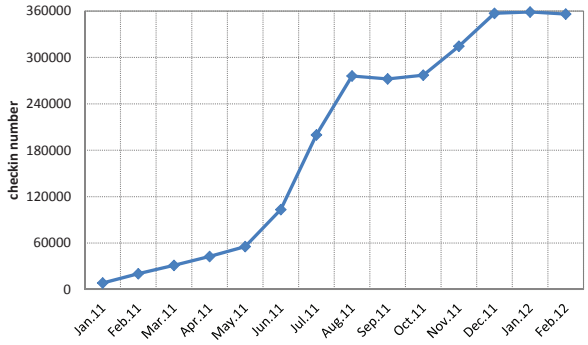


Figure 2. the Time-Checkin Distribution

Location-Checkin Distribution

The 80/20 rule is probably one of the most powerful ideas, which is universally applicable in the daily activities of our society. The underlying idea is that only a small part, about 20%, is important in a group, whereas the other 80% is trivial. In this paper, wo show that only a small number of locations, less than 20%, are popular among over 80% people, and many less popular locations have been visited by only a few people.

In this paper, we study the problem of ranking on LBSN in some location, and the locations in our checkin records are the name of the concrete geographic locations, such as POIs, districts and cities, instead of geographic coordinates. A POI is a small location such as restaurant, cinema or square, a district is bigger location that contains many small POIs, and a city is the same as our intuition. We analyze the location-checkin distribution in our dataset, and the details can be seen in figure 3. As can be seen from the figure that, about 20% POIs have been visited by nearly 80% people. As for districts or cities, about 10% of them have been visited by more than 98% people. This is because only a number of districts or cities are prosperous, and they attract more people. Table 2 lists the top-10 cities that people usually checkin in China.

Table 2. Top-10 Cities

1	2	3	4	5
Shanghai	Beijing	Guangzhou	Tianjin	Nanjing
6	7	8	9	10
Shenzhen	Hangzhou	Shuzhou	Wuhan	Xi'an

Degree-Checkin Distribution

The checkins in our dataset are also related to the degrees of people. Figure 4 illustrates the distribution of checkins according to people' degrees. The in-degree of a people is the number of people that follow him or her, the out-degree means the number of people he or she follows, and the degree is the sum of in-degree and out-degree. As can be seen from figure 4 that, the distribution of checkins is mainly decided by the in-degree. The reason is that when people have

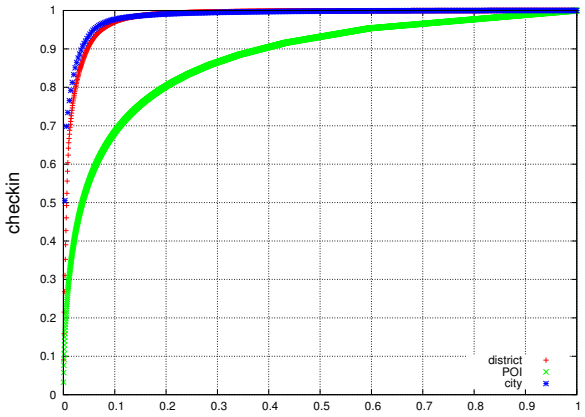


Figure 3. the Location-Checkin Distribution

more checkin records and comment more about those corresponding locations, they will attract more followers.

EXPERIMENTS

In this section, we present the results of the experiments that we have done to test the performance of our algorithm.

Experimental Setup

MapReduce [8], proposed by Google, is a programming model for processing huge amounts of data in parallel using a large number of commodity machines, and its open-source implementation is Hadoop[4]. By automatically handling the lower level issues, such as job distribution, data storage and fault tolerance, it allows programmers without any experience in parallel and distributed systems to easily utilize the resources of a large distributed system.

In this paper, we implement the LBSNRank algorithm in Java on top of the Hadoop platform. Our experiments are executed on a cluster of 20 nodes, where each node is a commodity machine with a 2.16GHz Intel Core 2 Duo CPU and 1GB of RAM, running CentOS v6.0. In order to demonstrate the robustness of our algorithm and to show its performance on realistic data, we present experiments with the Dianping dataset that we have crawled. Details of the dataset can be seen in section .

Experimental Results

In this section, several experiments are performed to compare the proposed LBSNRank algorithm with the traditional personalized PageRank.

Efficiency Evaluation

It needs several days to compute personalized PageRank with respect to all districts. Even if this is tolerable, then returning popular POIs or influential persons several months ago would be meaningless. There are two methods that can reduce the execution time, i.e., preprocessing a number of PageRank scores offline and compressing the graph with smaller subgraph. The LBSNRank algorithm is a composition of the two, figure 5 illustrates the comparison of execution time.

[4]hadoop.apache.org/common/docs/r0.16.4/hdfs_design.html

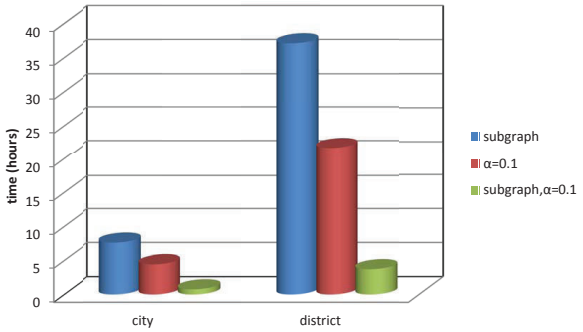


Figure 5. Comparison of Execution Time, where $\alpha = 0.1$ is the location ratio that we preprocessed offline

Though we already computed some popular personalized PageRank scores offline, there are still many personalized PageRank scores to compute online when queries are not hit. If the online computation takes much time, then a lot of queries would be blocked, and the throughput of the system would decrease. But as the locations become less popular, the people that visit them become fewer and fewer, so the subgraph of a LBSN becomes smaller and smaller, and we need less time to compute personalized PageRank scores online. Details can be seen in figure 6. Because the hadoop platform need some time to start-up the virtual machines and to process the input, the execution time cannot be small enough, and it tends to be a constant.

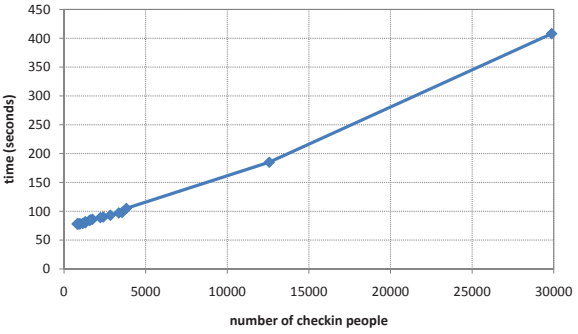
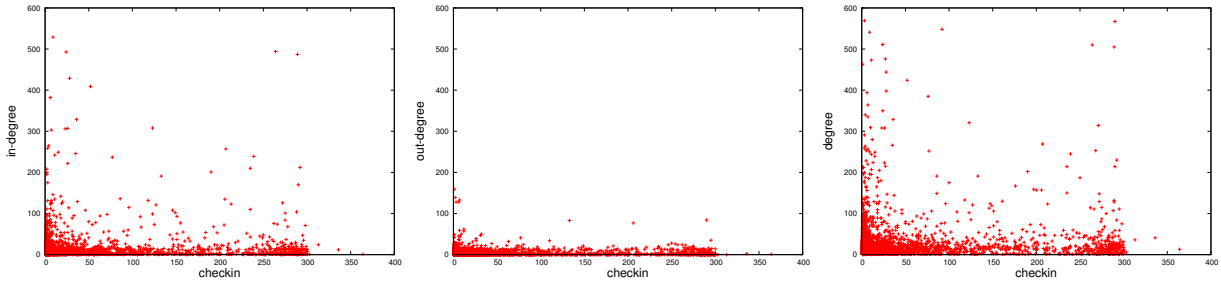


Figure 6. the Subgraph-Time Distribution

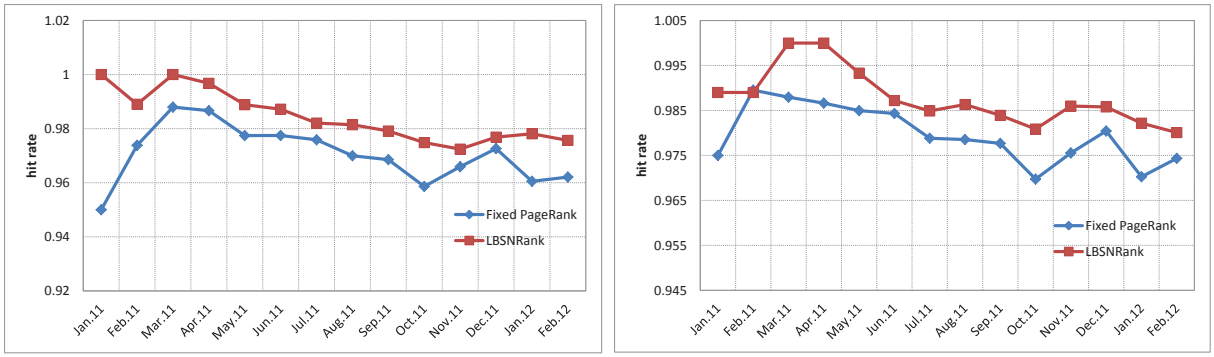
Hit Rate Evaluation

The hit rate of query is the ratio of queries that require results that have been preprocessed. When a query is issued by a user, the system seeks and returns the highest ranking results for that query. Because we only preprocess a number of personalized PageRank, there are some queries that require personalized PageRank that we haven't preprocessed yet. If we require a location that hasn't been preprocessed, then it will take more time to compute it online, and this will degrade the efficiency of the system. Therefore, if we improve the hit rate, we would save more time and resources, and hence improve the efficiency of the system. Though HubRank [6] computes some fixed personalized PageRank carefully, the



(a) in-degree (b) out-degree (c) degree

Figure 4. the Degree-Checkin Distribution



(a) city, $\alpha = 0.1$ (b) district, $\alpha = 0.1$

Figure 7. Hit Rate Comparison, where $\alpha = 0.1$ is the location ratio that we preprocessed offline

popularity of location changes quickly, which makes the hit rate of query even lower. In this experiment, we choose 10% ($\alpha = 0.1$) locations according to their popularity in the whole dataset to preprocess offline. In contrast to HubRank's fixed set, the proposed LBSNRank algorithm adjusts the location set that will be computed offline. In our experiments, we choose 35 ($\alpha = 0.1$) most popular locations as preprocessed location set in each month. In order to testify the accuracy of the LBSNRank algorithm, we choose 1% records of the following moth as test queries, and study the hit rate of query, details can be seen in figure 7. From the figure we can see that the LBSNRank algorithm has a better hit rate than the fixed personalized PageRank.

CONCLUSION AND FUTURE WORK

We study the problem of LBSNRank, i.e., the personalized PageRank on location-based social networks which are based on users' checkin histories. We rank locations based on their popularity, and for a specific location, we rank users based on their relationships and checkin records. As users in location-based social networks change their locations frequently, the rankings of locations and personalized PageRank scores of users change frequently as well. As MapReduce programming model becomes increasingly popular, we evaluate our experiments on its open-source implementation Hadoop. In order to validate our LBSNRank algorithm on real dataset, we have crawled a dataset from the Dianping website, and also analyzed some characteristics of users' checkin records. Experiments on this real dataset show that our LBSNRank algorithm is not only efficient, but also improves the hit rate of query.

In this paper, we precompute rankings with respect to popular locations, and thus we can answer users' queries timely with these results. However, it takes several days to rank all the districts in our experiments, and as for POIs, the execution time would be unacceptable. Though we precompute only a few of them offline, the computation still needs much time. Therefore, how to answer users' queries timely with up-to-date results would be our future work.

REFERENCES

1. K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 45(2), 2007.

2. B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *Proceedings of the 2011 international conference on Management of data*, pages 973–984. ACM, 2011.

3. B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

4. A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *Proceedings of the Thirtieth international*

conference on Very large data bases-Volume 30, pages 564–575. VLDB Endowment, 2004.

5. D. Cai, X. He, J.-R. Wen, and W.-Y. Ma. Block-level link analysis. In *The 27th ACM/SIGIR International Symposium on Information Retrieval*, pages 440–447, New York, NY, USA, 2004. ACM.

6. S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*, pages 571–580. ACM, 2007.

7. S. Chakrabarti, M. Joshi, K. Punera, and D. Pennock. The structure of broad topics on the web. In *Proceedings of the 11th international conference on World Wide Web*, pages 251–262. ACM, 2002.

8. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

9. D. Fogaras and B. Rácz. Towards scaling fully personalized pagerank. *Algorithms and Models for the Web-Graph*, pages 105–117, 2004.

10. D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

11. T. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002.

12. T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing pagerank. Technical Report 1999-31, Stanford University, 2003.

13. H. Hwang, A. Balmin, B. Reinwald, and E. Nijkamp. Binrank: Scaling dynamic authority-based search using materialized subgraphs. *Knowledge and Data Engineering, IEEE Transactions on*, 22(8):1176–1190, 2010.

14. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

15. S. Kong and L. Feng. A tweet-centric approach for topic-specific author ranking in micro-blog. In *Proceedings of the 7th international conference on Advanced Data Mining and Applications - Volume Part I*, ADMA'11, pages 138–151, Berlin, Heidelberg, 2011. Springer-Verlag.

16. N. Litvak. Monte carlo methods of pagerank computation. *Department of Applied Mathematics, University of Twente*, 2004.

17. L. Page, S. Brin, R. Motwani, and Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

18. D. Pennock, G. Flake, S. Lawrence, E. Glover, and C. Giles. Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the National Academy of Sciences*, 99(8):5207, 2002.

19. M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. *Advances in neural information processing systems*, 14:1441–1448, 2002.

20. D. Romero, W. Galuba, S. Asur, and B. Huberman. Influence and passivity in social media. *Machine Learning and Knowledge Discovery in Databases*, 6913:18–33, 2011.

21. A. Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011.

22. S. Scellato, C. Mascolo, M. Musolesi, and V. Latora. Distance matters: geo-social metrics for online social networks. In *WOSN'10*, Berkeley, CA, USA, 2010. USENIX Association.

23. S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo. Socio-spatial properties of online location-based social networks. *Proceedings of ICWSM*, 11:329–336, 2011.

24. D. Tunkelang. A twitter analog to pagerank. *The Noisy Channel*, 2009.

25. J. Weng, E. Lim, J. Jiang, and Q. He. Twitterrank: finding topic-sensitive influential twitterers. In *In Proceedings of the Third ACM International Conference on Web Search and Data Mining*, 2010.

26. G. Xue, Q. Yang, H. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *The 28th ACM/SIGIR International Symposium on Information Retrieval*, pages 186–193, New York, NY, USA, 2005. ACM.

27. M. Zhang, C. Sun, and W. Liu. Identifying influential users of micro-blogging services: A dynamic action-based network approach. In *PACIS Proceedings*, 2011.

28. Y. Zheng. Location-based social networks: Users. In Y. Zheng and X. Zhou, editors, *Computing with Spatial Trajectories*, pages 243–276. Springer New York, 2011.

29. Y. Zheng and X. Xie. Learning location correlation from gps trajectories. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, pages 27–32. Ieee, 2010.

30. Y. Zheng and X. Xie. Learning travel recommendations from user-generated gps traces. *ACM Transaction on Intelligent Systems and Technology (ACM TIST)*, 2(1):2:1–2:29, Jan. 2011.

31. Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800. ACM, 2009.