Fast Computation of All-pairs Random Walk on Large Graphs

Sunchan Park' and Sang-goo Lee',

' School of Computer Science and Engineering, Seoul National University, Seoul, Korea
{baksalchan, sglee}@europa.snu.ac.kr

Abstract. In this paper, we propose a novel strategy for fast computation of random walk probability for all node pairs on large graphs. Since computation of random walk includes a large number of random accesses to adjacency lists, it is time consuming task when whole graph cannot be loaded into main memory. We devise an efficient random walk computation algorithm for large graphs by reducing the number of random accesses to adjacency lists. We reduced the number of random accesses by reusing pre-computed results of other similar nodes. We also provide some discussions about related issues.

Keywords: graph, random walk

1 Introduction

Recently utilizing heterogeneous graphs gets attentions by researchers in knowledge management and recommendation fields. PathSim[1] and HeteSim[2] are the recent result of the studies that shared the point of view. Computation of random walk probability is the basic and core task of heterogeneous graph analysis.

For example, the probability that a random walker starting from some user node goes through the path " $User \xrightarrow{Like} Movie \xrightarrow{Like^{-1}} User$ » and reaches some other user node means "the probability that some user likes the movies that are liked by the some other user". This result can be used as a similarity measure between users. In the similar way, the random walk probability going through the path " $User \xrightarrow{Neighbor} User \xrightarrow{L} Movie$" can be utilized as a relevance measure of movies to users. Random walk computation can be viewed as matrix-matrix multiplication, and there exist many efficient implementations performing the task. However, they perform the task after loading whole matrices on main memory, and it is not possible to load large graphs like social networks on main memory. Thus we need other method considering non-memory based environments.

In this paper, we propose a novel random walk computation strategy for large graphs. The random walk computation includes a large number of random accesses to adjacency lists, and the random access count is one major factor of the performance. Our novel algorithm can perform the random walk computation with reduced number of random access count by reusing pre-computed result of similar nodes.

The remained part of paper is organized as follows; we describe the detail of our novel algorithm to solve the problem in chapter 2. We also provide some related issues at the end of chapter, and we conclude this paper by chapter 3.

2 Strategy for All Pairs Random Walk Computation

In this chapter, we provide the formal definition of all-pairs random walk computation and describe the efficient algorithm to solve it. Firstly, all-pairs random walk computation problem is defined as follows.

Definition 1. When a heterogeneous graph *G.(V, E),* and a path P = $N_1 \xrightarrow{E_1} N_2 \xrightarrow{E_2} \cdots \xrightarrow{E_k} N_{k+1}$, where N_i is a node type and E_i is a edge type, the all pair random walk computation problem is defined as computing the probability that the random walker starting from the node n_1 reaches the other node n_2 by going through the path P for all node pairs $(n_1, n_2) \in V \times V$. it can be alternatively defined as computing the probability vector *r(n)* representing the probability that the random walker starting from the node *n* reaches all other nodes through the path *P* for all node $n \in V$.

Solving the all pairs random walk computation problem is exactly same as the matrix-matrix multiplication of matrices that represent the each transition in the path. Consequently, this problem can be solved with matrix-matrix multiplication. However, because of the problems that we stated the previous chapter, the current matrix-matrix multiplication method cannot be directly applied to large graphs. The other problem is that large graph data is stored in the form of a massive adjacency list, not a sparse matrices format for a mathematical computation.

Our main idea is reusing the pre-computed results of the similar nodes. If the adjacency list of one node is similar to the one of some other node, we can expect that the random walk computation result from each node is also similar to each other. We utilize the intuition.

Let's start with the case of computing 2-step random walk following any type of node and edge. If we Assume that each adjacency list of two nodes n_i, and n_1 differs only one node from each other, we can say that $Adj(n_i) = \{n_1, n_2, \ldots, n_k, n_{k,i}\}$, and $Adj(n_j) = \{n_1, n_2, \ldots, n_k, n'_{k+1}\}$, where *Adj(n)* is the adjacency list of node *n.* If we have *r(n,),* we can compute $r(n_j)$ using *r(n,)* by the following equation;

$$r(n_j) = \frac{1}{|Adj(n_j)|}\left(|Adj(n_i)|\, r(n_i) - a(n_{k+1}) + a(n'_{k+1})\right) \quad (1)$$

a(n) represents the normalized vector representation of the adjacency list of a node *n.* By using the equation (1), we can compute $r(n_j)$ without accessing all adjacency lists of *n; s* adjacent node. Random access count required for computing $r(n_j)$ is reduced from $|Adj(n_j)| + 1$ to 3. If the degree of nodes are larger, this strategy speed up more

the random walk computation. In general case, we can rewrite the above equation as follows.

$$L^{I} \frac{I}{\,} Adj(n_i)1(_{r(n,)} \frac{1}{\,} \quad Ni_{a(nk)})_{+} \frac{1}{1\ Adj(n_i)1Li} \quad Ni_{ao,k)} \qquad \textbf{(2)}$$

In the equation (2), n_k represents the node that is included in *Adj(n;)* but not *Adj(ni),* and n_{L_c}' represents the node that is included in *Adj(n3)* but not *Adj(nt).* In general case, the random access count is 1 + $a(n_k)$ + $a(n'_k)$, and the speed up can be assessed as (1 + IA *dj* *(ni)l) 1 (1 + E a (n k) + a(n'k)).* The algorithm computing all pair random walk is described in Fig. 1.

2 step all-pairs random walk computation *(G, P)*
begin
  foreach node *n* in *G* **do**
    n' getComputedSimilarNode(n);
    if *n' null* **then** compute *r(n)* from *r(n')* using equation (2);
    else compute *r(n)* by referring *Adj(a)* for each node *a* E *Adj(n);*
  end
end

Fig. 1. Algorithm for two-step random walk computation

Finding pre-computed similar nodes can be done in various ways. It is important implementation issue affecting performance. We can use fast node clustering algorithms like METIS [3] or MinHash[4] to judge similarity between adjacency lists.

Utilizing parallel computation capability is also important issue. This strategy is inherently sequential. However, we can parallelize the algorithm by split it into two phases: 1) seed computing phase and 2) seed referencing phase. In seed computing phase, we compute the random walk results from seed nodes, and compute the random walk result from other nodes by reusing the result from the first phase. Internal process of each phase can be performed in parallel fashion. We can make full use of parallelism by setting the number of seeds as the degree of parallelism.

We discussed 2-step random walk computation without type constraint so far. However, our strategy can be easily extended to multi-step random walk computation since two-step computation results can be seen as another 1-step relationship. We can compute any length of random walk probability by iteratively applying the previous algorithm. Type constraints also can be easily applied by using type-filtered adjacency lists.

3 Conclusion

In this paper, we propose the efficient strategy for all pair random walk computation on large graphs. We optimize the random walk computation by reusing pre-computed result of other nodes that have similar adjacency lists. We plan to implement the algorithm and evaluate the effectiveness of our strategy.

4 Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20120005695).

References

1. Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, Tianyi Wu: PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. PVLDB. Vol. 4. Issue. 11. pp. 992--1003. (2011)
2. Chuan Shi, Xiangnan Kong, Philip S. Yu, Sihong Xie, and Bin Wu. Relevance Search in Heterogeneous Networks. In: 15th International Conference on Extending Database Technology, Berlin, Germany, 2012.
3. George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal. Vol. 20. Issue. 1. (1998)
4. Kathy Macropol, Ambuj K. Singh: Scalable Discovery of Best Clusters on Large Graphs. PVLDB. Vol. 3. Issue. 1. pp. 693-702. (2010)
5. Sangkeun Lee, Sang-I1 Song, Minsuk Kahng, Dongjoo Lee and Sang-Goo Lee: Random Walk based Entity Ranking on Graph for Multidimensional Recommendation. RecSys 2011.