DATA130045

Graph Data Management and Mining

Tim Walker

2022-01-18

Final Project
SimRank Enhanced by Semantic

Abstract

SimRank is a widely used and popular measurement to evaluate the similarity among the vertices. However, with more and more graphs involves some of the labels that contain semantic message, the original SimRank might fail to deal with some of the queries. In our final report we will present comprehensive explanation about our idea and some of the experiments done by us.

1 Introduction

Estimating node similarity with SimRank is a well-established technique that we have several high efficient ways to calculate the approximate SimRank (or accurate one). However, the traditional SimRank only consider the structure of the graph itself, but neglect the labels in the informative networks, which could be a great loss of information on the query of such networks. Therefore, some papers have already done some research on this field, presented different solutions for the semantic-enhanced SimRank computation.
As we know most of the SimRank-based top-k queries on large graph use the randomwalk-based method, and the variants of the method have been proved efficient and could handle most of the cases. The READS method is a pretty efficient method to calculate the SimRank on graphs without label, it is also used in LSimRank to boost the calculation of semantic-enhanced SimRank. However, no matter the SemSim or LSimRank, the way they transfer information during random walk is an inseparable opperation. For instance, the core equation of SemSim is

$$\frac{\mathrm{sem}(u,v)\cdot c}{N_{u,v}}\sum_{i}^{|I(u)||I(v)|}\sum_{j}\mathrm{sim}\left(I_i(u),I_j(v)\right)\cdot W\left(I_i(u),u\right)\cdot W\left(I_j(v),v\right) \tag{1}$$

The part $\mathrm{sem}(u,v)$ increase the difficulty of randomwalk, which, in the graph similarity query problem neglect the labels, they only need to store the $Next_i$ index, which is $O(n)$ storage complexity for each round of simulation. But for SemSim or LSimRank, if they adopt the framework of READS method (or other approximation methods), the path of each randomwalk must be stored for the similarity computation. Which not only increase the storage complexity, but also increase the query time as they need to backtrace the path in each round of offline simulation.
To handle the above situation, as the weight of the edge in equation (1) could be added into READS with simple modify of the origin algorithm, we want to make use of the weight of edges in the graph to transfer the semantic information. Therefore, we think the paper of Semantic Proximity Search might help us with forming the proper edge weight.

2 Offline Phase

2.1 Metagraph Matching

We hinge on the novel insight that different semantic classes can often construct structures in common, namely metagraph. Based on the previous method called GRAMI algorithm, the metagraph

patterns could be generated automatically according to frequent pattern mining.
Then we propose a supervised approach that identifies the characteristic metagraph, in practice, we learn a weight for each metagraph to quantify how well it can stand the specific class.

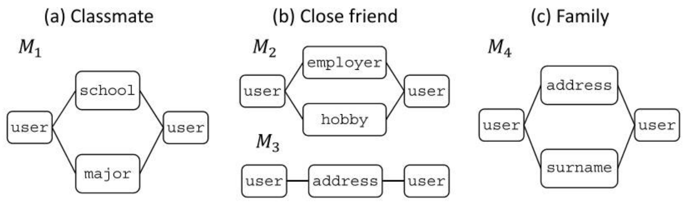

Figure 1: Possible Toy Metagraphs for some Classes

After implementing this, it comes to metagraph matching. While it is proved that computing the instances of a metagraph is highly costly, which is equivalent to solving a NP-hard subgraph matching problem, we shall work on another clever way that re-uses "symmetic" characteristic. As a result, we can avoid redundancy and substantially improve the efficiency of metagraph matching.
In order to know whether two nodes share a characteristic metagraph pattern for the specific type of proximity, it is crucial to identify subgraph on the whole object graph that match instances of any metagraph.
Before leading to our weight formula, here're some notations denotes the number of instances of each metagraph that matches between nodes.

$$\mathbf{m}_{xy}[i] \triangleq \left| \{S \in \mathcal{I}(M_i) \mid \text{ ContainsSym } (S, x, y)\} \right| \tag{2}$$

$$\mathbf{m}_{x}[i] \triangleq \left| \{S \in \mathcal{I}(M_i) \mid \exists_{y \neq x} \text{ ContainsSym } (S, x, y)\} \right| \tag{3}$$

2.2 TF-IDF-like Formula

As for the original Metagraph Matching, we could define different kinds of relations, and distinguished by different pattern and pattern weight, we could easily compute the similarity between nodes. But in many cases, we need to find the similarity between nodes without specific relation, which means, the relation itself also could be important or not. Therefore we make use of the TF-IDF method in LSA and present the importance of each in-neighbor of node x in the following way:

$$\sum_{\omega_i} \frac{m_{xy}^T \omega_i}{m_x^T \omega_i + m_y^T \omega_i} \log \frac{|E| e^T \omega_i}{\sum m_{xy}^T \omega_i} \tag{4}$$

Where m_x and m_{xy} means metagraph vector for node x and node pair x, y, respectively. w indicates characteristic weight vector.
After normalization, we could sample each in-neighbor of node x with specific probability to specify its importance.

2.3 READS Offline Indexing

In the problem concerning simple SimRank query without label, the READS framework provides a way to construct efficiency index for online query, and it only require small amount of storage. The basic idea is, in the original randomwalk method, we have

$$s(u, v) = \sum_{i=1}^{t} \Pr\left(f\left(\pi_u, \pi_v\right) = i\right) \times c^i \tag{5}$$

Thus, to calculate the approximation of SimRank, we need to store the length of two randomwalk before their first meet. To improve it, READS use different way to sample the in-neighbor of a node, which could be present as follow

a	$\Pr\left(u_1 = v\right) = \frac{1}{\lvert \mathrm{In}(u_0) \rvert}$ for $v \in \mathrm{In}\left(u_0\right)$
b	$\Pr\left(u_{i+1} = v\right) = \frac{\sqrt{c}}{\lvert \mathrm{In}(u_i) \rvert}$ for $v \in \mathrm{In}\left(u_i\right), 0 < i < t$

(6)

With this kind of sampling technique, the randomwalk might stop at certain step, which means it contains the probability that two path will not meet within the step limitation. Therefore, we have the following equation

$$s(u,v) = \sum_{i=1}^{t} \Pr^{SA}\left(f\left(\pi_u, \pi_v\right) = i\right) \times c \tag{7}$$

From which we could see that no matter how many steps does the simulation takes for two reverse randomwalk to meet, they all have the same amount of contribution to the SimRank. That is to say, to calculate the approximation of SimRank of two nodes u, v, all we need to do is simply counting the number of time that the two reverse randomwalk starting from u, v meet each other in the total r rounds, and each of them has the contribution of c/r. We could present this outcome with the following equation

$$\begin{aligned} s(u,v) &= \sum_{i=1}^{t} \Pr^{SA}\left(f\left(\pi_u, \pi_v\right) = i\right) \times c \\ &\approx \widetilde{\Pr}^{SA}\left(f\left(\pi_u, \pi_v\right) \neq \infty\right) \\ &= \left|\left\{ j \leq r \mid f\left(\pi_u^j, \pi_v^j\right) \neq \infty \right\}\right| / r \end{aligned} \tag{8}$$

With the above outcome, we could deploy the corresponding simulation. In each round of simulation, we start from leaf nodes, i.e. all the nodes on the graph, and deploy reverse randomwalk from all the leaf nodes with the above rules. If two randonwalk meet each other, they will simply merge to each other and continue doing reverse randomwalk as one branch instead of two separate branches. A branch of reverse randomwalk will stop once it has the length of t, i.e. the limitation of maximum randomwalk steps, or it stops as the above rules. Then we will attain a forest, which to check whether two randomwalks meet each other, we just need to check if the two nodes on the same tree, and we could form a index which is handy and easy for query. Here is an example

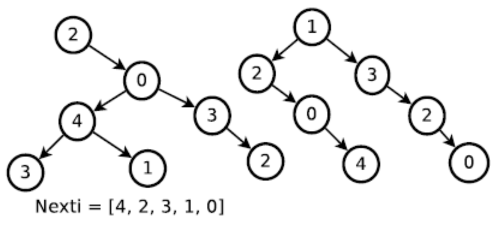

Figure 2: READS Example

The i^{th} element of $Nexti$ represents the right sibling of leaf node i, if i is the rightmost leaf node in the tree, then $Nexti[i]$ represents the leftmost node in the tree. In this case, we have node 4's right sibling is 0, so $Nexti[4] = 0$. Thus, each round of simulation only takes $O(n)$ of storage, with r rounds of simulation we only need $O(rn)$ of storage (The effect of c will make all the tree really short, thus we could take the height of tree as a small constant).

2.4 REAEDS with Weight on the Edges

To make use of the information of edge weight, we just need to adjust the sampling rule in (4) in the following way

$$\begin{array}{|c|l|}\hline \mathbf{a}' & \Pr\left(u_1 = v\right) = \frac{W(v,u_0)}{W(u_0)} \text{ for } v \in \operatorname{In}\left(u_0\right) \\ \hline \mathbf{b}' & \Pr\left(u_{i+1} = v\right) = \frac{W(v,u_i)}{W(u_i)} \text{ for } v \in \operatorname{In}\left(u_i\right), 0 < i < t \\ \hline \end{array} \tag{9}$$

Where $W(u, v)$ stands for the weight of edge (u, v), and $W(u) = \sum\limits_{v \in \operatorname{In}(u)} W(v, u)$

3 Online Phase

The Online Phase is simple for READS framework. To calculate one-to-all query, using the outcome of the former chapter, we have the following algorithm

Algorithm 1 One-to-all Query

```
Input: Next, u
Output: $s_1(u, *)$
1: for i = 1:r do
2:     $v \leftarrow Next_i[u]$
3:     while $v \neq u$ do
4:         $s_1(u, v) += c/r$
5:         $v \leftarrow Next_i[v]$
6:     end while
7: end for
8: $s_1(u, u) \leftarrow 1$ Return: $s_1(u, *)$
```

4 Experiments

We implemented our ideas above in order to demonstrate two goals. First, our implementation can return query results as quickly as possible. Second, the output can be validated to show that sematic information truly improves the accuracy.

4.1 Dataset

To test our algorithm's efficiency, we use the following three datasets, one is the famous LinkedIn dataset with sematic labels, one is Facebook, and the other is self-constructed social network, which is taken from the real classmates around us.

	#Nodes	#Edges	#Types	#Metagraphs	#Queries
LinkedIn	65925	220812	4	164	172
Facebook	5025	100356	10	954	904
FudanDS	67	712	3	10	10

We conducted extensive expriments on the real-world datasets LinkedIn, Facebook and FudanDS. All datasets contain features of various types. In particular, LinkedIn included user, employer, location and college. Facebook included user, major, degree, school, hometown, surname, location, employer, work-loaction, work-project and others. FudanDS included research direction, dormitory and social cycle.

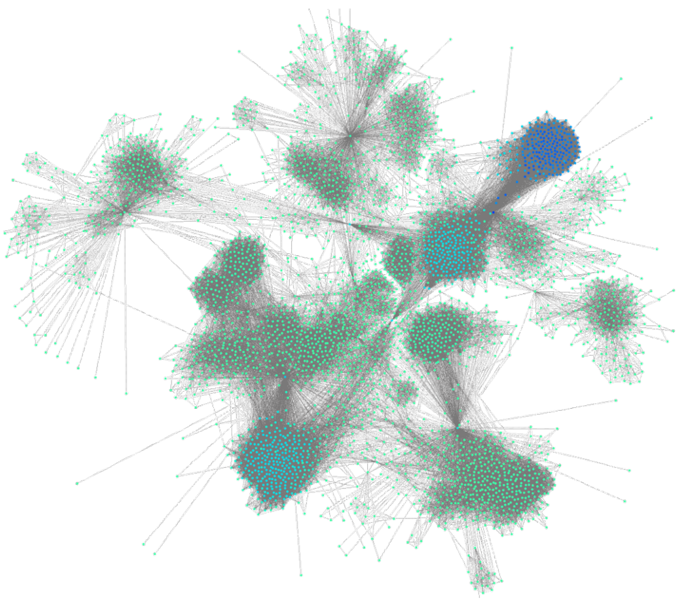

Figure 3: Facebook Dataset Visualization

4.2 Computing Weight

In this part, We applied preprocessing algorithm GRAMI on each graph to detect the set of metagraphs, and according to Metagraph-based Search, we only retained the symmetric ones. Then we train each edge's weight using part of Metagraph-based Search by replacing the original core formula with our improved TF-IDF-like weight formula, by this way we extracted all the features and obtained the corresponding scores, and the linear weighted summations are the edges' weights.

4.3 Generating READS

From the advanced READS algorithm above, the READS tree grows with the probabilities related to the linked edges' weights of the node, which are normalized beforehand. And the offline work only stored the next-list for each simulation, as known as $O(rn)$ space complexity.
After this, online queries will returns results from One-to-all algorithm. And our experiment results are as following.

4.4 Results

We illustrate the average matching accuracy for LinkedIn and Facebook per query sample batch in Figure 4, where the size of each sample is 87 and 246, respectively. It's obvious that our algorithm has a better behavior on Facebook significantly, and the reason is also obvious. Facebook dataset has 10 types in total which includes details like work-project, while LinkedIn dataset only covers 4 rough types, that is to say Facebook dataset's sematic label has more tiny grain, and thus providing more credible evidence in the decision of weight scores.

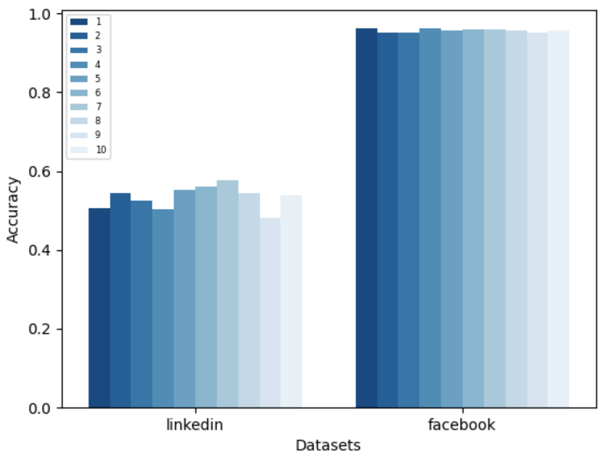

Figure 4: Query accuracy results for LinkedIn and Facebook

The first graph shows the similarity without semantic information for node 17, while the second shows the the similarity with semantic weight for the same node. As for the top 5 simrank nodes, original READS algorithm results to node 61,0,27,15 and 55, however, node 0 and node 27 both have nothing in common with node 17 at all in reality, actually they live in social circles without any intersection. On the other hand, our algorithm presents better results which are node 42,38,15,32 and 65, in which node 42 is truly considered as the most similar one with node 17 in our anticipation. On the contrary, the orighial READS gives 6,41,57,56 and 8 as top 5 dissimilar nodes for node 17, but unfortunately node 6 and node 57 are both like-minded friends with node 17. While our algorithm gives node 53,24,21,26 and 35, which are strangers with id 17 and totally different with node 17, reasonable. Though the man-made dataset may have some inevitable mistakes, generally our modified algorithm always shows a better ranking on similarity.

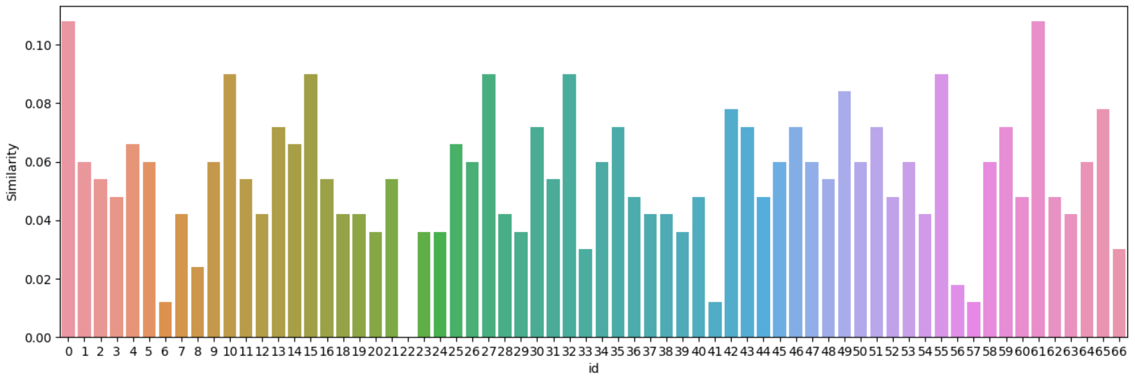

Figure 5: FudanDS similarity sample (without semantic)

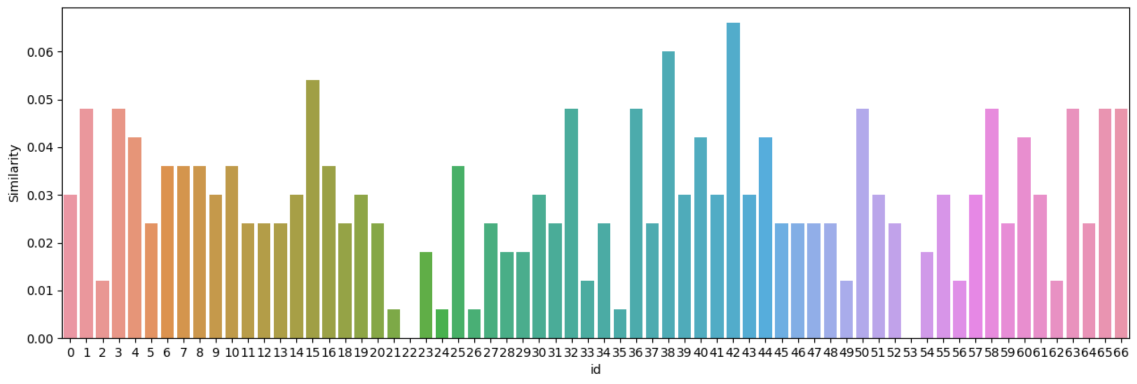

Figure 6: FudanDS similarity sample (semantic)

5 Conclusion

To put it into a nutshell, MARS (Metagraph-based semAntic-boosting Reads Simrank) algorithm takes advantage of the semantic weighted information generated from metagraph and the linear space complexity inspired from READS. Furthermore, we improve the weighted calculation formula with the idea migrated from TF-IDF in LSA, which considers both semantic label's similarity and global semantic specificity.

References

[1] Youngmann, B., Milo, T., Somech, A.: Boosting simrank with semantics. In: EDBT, pp. 37–48 (2019)

[2] Wang, X., Zhang, R., Lee, Y.-K., Sun, L., & Moon, Y.-S. (Eds.). (2020). Web and Big Data. Lecture Notes in Computer Science. doi:10.1007/978-3-030-60259-8

[3] Fang, Y., Lin, W., Zheng, V. W., Wu, M., Chang, K. C.-C., & Li, X.-L. (2016). Semantic proximity search on graphs with metagraph-based learning. 2016 IEEE 32nd International Conference on Data Engineering (ICDE). doi:10.1109/icde.2016.7498247

[4] Jiang, M., Fu, A. W.-C., & Wong, R. C.-W. (2017). READS. Proceedings of the VLDB Endowment, 10(9), 937–948. doi:10.14778/3099622.3099625