User-Guided Meta-Path Based Framework in Heterogeneous Networks

Hilfi Madari Alkaff, Efe Karakus
{alkaff2, karakus1}@illinois.edu

Abstract

With the advent of the big-data era, there have been many large-scale heterogeneous information network that consist of multi-typed, interconnected objects such as social media networks or news networks. Many companies have been established to unearth patterns in such a massive dataset. One of the operations that people have been interested is performing is similarity search. However, most existing similarity search frameworks have been unable to support user queries in real-time. Furthermore, users are not able to perform OLAP operations efficiently on the data. As a result, it is not possible for user to compute similarity measures between two objects under the contexts that the user desires. In this paper, we propose similarity search that support both online and OLAP queries for user.

1 Introduction

In the current era of big data, an enormous amount of information is being generated in real-time. It is important that we are able to navigate these massive datasets to understand what is currently going on. We would like to specifically focus on news data for a number of reasons. Firstly, there is an abundance of available news data that we could analyze. Secondly, it is possible to categorize news data hierarchically. An example of such hierarchy is the country, followed by state, district, and so on. We should be able to seamlessly navigate between different level in the hierarchy to retrieve the relevant information. Last but not least we choose news data because it allows us to be more aware of the current events that are happening around us. As such, we would like a platform that allow us to answer questions such as 'Who are the key politicians for healthcare reforms?', 'How did the opinion of Barack Obama change over time for education?' and so on.

Similarity search has been extensively studied in the past. At first, it was applied to only categorical and numerical data types in relational data. Then, there have been works that tries to leverage link information in networks. Most of these studies focus on homogeneous or bipartite networks [4, 2, 7]. However, these similarity measures disregard the subtlety of different types among objects and links which are present in heterogeneous information network. More recently, a meta-path based similarity framework, PathSim [5], has been proposed that takes into account different linkage in the network.

However, there are several shortcomings in the PathSim design. Firstly, the operations to compute the meta-path are computationally expensive and consume a lot of memories since it is achieved through matrix multiplications. Secondly, it does not allow users to perform any OLAP operations in the dataset. This is important because a user might be interested in the similarity of two objects under a specified context only. Thirdly, the original PathSim framework also does not allow user to provide hints in the type of meta-paths that a similarity search query should return. For instance, a user might be interested in a similarity search between two objects that does not go into a specific node in the graph. Finally, the meta-paths returned to the user might be hard to understand. For example, writing *co-author* instead of $author \rightsquigarrow paper \rightsquigarrow author$ is a lot more relevant to the user.

In this paper, we propose HPATHSIM, which is built on top of PathSim, that addresses the aforementioned shortcomings of PathSim. We address the first challenge by building a hash-table, METAPATH-TABLE, whose key is a pair of node while the value is the meta-paths between the pair. At first, the hash-table is empty. But as user queries for similarity measures on two nodes in the graph, the meta-paths found during the computation will be cached in the hash-table. This allows for faster indexing if the user performs the same query. For the second challenge, we address it by building a forest of hierarchical trees. Finally, the third challenge is addressed by keeping track of constraints that the user specifies in a hash-table, CONSTRAINT-TABLE. CONSTRAINT-TABLE will be checked whenever we found a candidate for meta-path between two objects to ensure that none of the constraints that the user specified is violated.

1.1 Paper Outline

This paper is structured as follows. We begin with the design of our framework (Section 2) followed by our implementation (Section 3). We then show the performance of our framework based on the DBLP and NYTimes dataset (Section 4). Finally, we discuss some extensions that could further be made to our framework and conclude (Section 5).

2 Design

In this section, we begin with describing the original PathSim framework in a greater detail. Then, we will delve into the changes that we made to the PathSim framework to implement a more efficient similarity search framework while being able to support hints and OLAP operations from the user.

2.1 PathSim

PathSim is developed to compute similarity measures in a heterogeneous information network which are logical networks that involve multiple typed vertices and multiple typed links denoting different relations (e.g., bibliographic network, news articles).

Figure 1 shows an example heterogeneous information network. In this figure, there are 4 possible types of entities, author, paper, venue and term, which are inter-connected while there are also 4 different type of edges that exist (i.e., "Writes", "Has", "Cited" and "Submitted To").

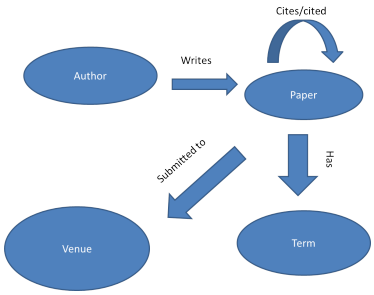

Figure 1: Bibliographic Network

PathSim strives to define similarity measures between vertices in heterogeneous information network using structural information and to answer top-k similarity search queries efficiently. It accomplishes this by developing a novel meta path-based framework.

As shown in Figure 1, entities can be connected via different connectivity paths. For instance, two authors can be connected via author-paper-author (i.e. co-authors) path or author-paper-venue-paper-author (i.e. the authors submitted both of their papers to the same conferences) path. Intuitively, the different paths represent a different similarity semantics. More formally, as written in the paper, the meta path can be described as follows:

Definition 1 ***Meta path.*** *A meta path $\mathcal{P}$ is a path defined on the graph on network schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $\mathcal{A}_1 \xrightarrow{R_1} \mathcal{A}_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} \mathcal{A}_{l+1}$ which defines a composite relation $\mathcal{R} = \mathcal{R}_1 \circ \mathcal{R}_2 \circ \cdots \circ \mathcal{R}_l$ where $\circ$ denotes the composition operator on relations.*

Although multiple similarities measure have been previously explored such as path counting or random walk-based similarity, they are biased to either popular entities and thus unable to capture the essence of peer similarity. For instance, if we would like to

find out about authors that are similar to an early PhD student that has published a few papers, the above measures will yield professors who happen to co-author with this particular student although what we desire are other PhD students. This has become the other motivation for PathSim. Using the concept of meta path above, Path-Sim defines the relationship between two entities as follow:

Definition 2 *PathSim: A meta path based similarity measure. Given a symmetric meta path $\mathcal{P}$, Path-Sim between two vertices of the same type x and y is:*

$$s(x,y) = \frac{2*|\{p_{x\rightsquigarrow x} : p_{x\rightsquigarrow y}\}|\in\mathcal{P}}{|\{p_{x\rightsquigarrow x} : p_{x\rightsquigarrow x}\}|\in\mathcal{P}+|\{p_{y\rightsquigarrow y} : p_{y\rightsquigarrow y}\}|\in\mathcal{P}}, \text{ where } p_{a\rightsquigarrow b} \text{ is the meta paths between } a \text{ and } b.$$

Intuitively, this indicates that if x is popular, but not y, the number of meta paths between x and itself will be much larger than the number of meta paths between x and y. Thus, when we are comparing a popular entity and an un-popular entity $s(x,y)$ will be low and vice versa. We need to have either both to be popular or un-popular so that $s(x,y)$ yield a considerable value.

2.2 hPathSim

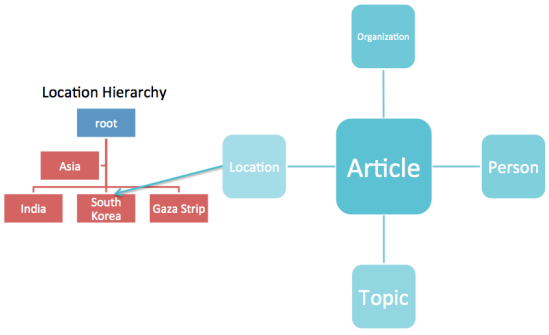

Figure 2: News Articles Network

There are three main improvements that we made to the PathSim framework. Firstly, even with the pruning algorithm that they introduce in the paper, the original PathSim framework still requires expensive matrix computations that are not feasible in an online setting. We introduce a better storage mechanism, MetaPath-Table, that allows faster materialization of the meta paths.

Secondly, a heterogeneous information network typically contains a hierarchy for its entities' dimensions. Figure reffig:news shows one of the hierarchies that exists in the news network. In this case, this particular article's location dimension corresponds to "South Korea" which is a child of the "Asia" category. Thus, we built our own tree data structure, Hierarchy Forest, to allow us to perform efficient OLAP operations on the dataset. Thirdly, we show how we are able to support user defined constraints in our dataset.

2.2.1 Storage Optimization

In hPathSim, we introduce a hash-table in which the key is a pair of node, $(\mathcal{N}_1, \mathcal{N}_2)$ while the value is a set of meta paths between the pair. At first, the hash-table is empty. But as user queries for similarity measures on two vertices in the graph, meta-paths found during the computation will be cached in the hash-table. This enables fast retrieval if the user issues the same queries in the future.

However, as noted in the original PathSim paper, it is impossible to cache all of the possible meta-paths in an enormous graph. The author of the paper calculated the storage size requirement of all the possible length-4 meta paths in the DBLP data set is more than 40GB. Thus, we use an underlying database in our framework to regularly swap meta paths in the cache to the disk if we are using too much memory. We use Least Recently Used (LRU) policy to determine which meta paths that need to be swapped to the disk due to its simplicity and its effectiveness in ensuring that it works.

The database is checked every time the user issues a similarity search query to see if meta paths between two vertices have been computed previously. If so, the meta paths will be swapped back to the cache again. Similar to our cache, the database will still be indexed by $(\mathcal{N}_1, \mathcal{N}_2)$ to allow fast retrieval.

2.2.2 Supporting OLAP Operations

An example of a location hierarchy could look like the following:

$$Urbana \rightsquigarrow USA \rightsquigarrow NorthAmerica \rightsquigarrow root$$

Thus, a user might want to perform the similarity search under the specified constraint (e.g. Find me the most similar person to Obama in Africa). This problem has not been addressed in the original PathSim framework.

To support OLAP operations, hPathSim builds a forest of hierarchical trees, the trees use HashTables in their core to allow for fast node retrieval. The roll-up and drill-down operations on the tree are implemented by a depth-first search (DFS).

The forest provides 5 key operations for each of the trees:

- *is_member(nodeTitle, queryID)*: Applies an iterative DFS to check if the node with id *queryID* is under the subtree of *nodeTitle*.
- *is_slice(nodeTitle, queryID)*: check whether the node with *nodeTitle* has the id *queryID*.
- *get_categories()*: return the tree titles in the forest.
- *get_children(nodeTitle)*: get all immediate children under the node *nodeTitle*.
- *get_parent(nodeTitle)*: get the parent of the node *nodeTitle*.

All the operations described above take $O(1)$ runtime due to the HashTable, except for *is_member* which can take $O(N)$ where N is the number of nodes under the specified hierarchy tree.

When the user needs to drill-down in our dataset, he needs to specify a category name and a category value. The drill-down operation is implemented such that for each node, we check if its category is a member of the pair (category name, category value). Since each vertex in the graph is only being looked at once in this operation, the running time for drill-down is $O(N)$ where N is the number of vertices in the current graph.

For the roll-up operation, the user will need to specify similar parameters. For this operation, we keep track of all of the nodes that belong to a particular category in the hierarchy trees and save them in a hash-table where the key is the category name in the hierarchy tree while the value is a set of nodes that belong to that category. We add all of these vertices together and then append them to the current graph. Thus, a vertex could appear in the hash-table multiple times depending on how far down from the root is his category. Thus, the running time of the roll-up operation is $O(hN)$, where h is the depth of the hierarchy tree while N is the number of vertices in the current graph.

2.2.3 User-Specified Constraints

Thirdly, we also allow user to specify constraints on the type of meta paths result that are used for computing the similarity score. There are two types of constraints that the user could specify. Firstly, a user could specify that meta paths between two vertices must go through a certain node or any nodes with a particular category type. Conversely, a user could also specify that meta paths between two vertices must not go through a certain node or any nodes with a particular category type. We choose these two types of constraints becaues we believe that they are intuitive for the users and will allow users to identify similarity between the two objects in a greater detail. These constraints are stored in a hash-table, Constraint-Table.

Similar to the original PathSim framework, the top-k paths used for the similarity search computation are discovered through a Breadth-First Search (BFS) algorithm. We based our algorithm off BFS due to its simplicity and guarantee to compute shortest path between two vertices.

Algorithm 1 shows the algorithm that we use to discover the top-k paths with user-specified constraints. The first *check* function is called every time a new partial path is explored. It ensures that nodes that

Algorithm 1 Constrained BFS Algorithm to Find Meta-Paths

```
1:  function FindPath(c, src, dst)
2:      q ← Queue()
3:      paths ← Set()
4:      q.enqueue(Array(src))
5:
6:      while not q.empty() do
7:          curPath ← q.dequeue()
8:          lastNode ← curPath.lastNode
9:          for n ← lastNode.neighbors do
10:             if check(n, c, "mustNotHave") then
11:                 continue
12:             else if n in curPath then
13:                 continue
14:             end if
15:
16:             newPath ← curPath.append(n)
17:             if n == dst then
18:                 if check(n, c, "mustHave") then
19:                     paths.add(newPath)
20:                 end if
21:             else
22:                 q.enqueue(newPath)
23:             end if
24:         end for
25:     end while
26:
27:     return paths
28: end function
```

user *does not want* to be added are being pruned from the search space to save computation time. The second *check* function is called only when a full candidate path between the source and destination is discovered. This ensures that all the nodes that the user *wants* exist in the path.

2.2.4 User-Defined Meta-Paths

Finally, the meta-paths returned by PathSim might be hard to read. For example, the meta-path *Article⇝Person: Coen Brothers ⇝ Article ⇝ Location: France ⇝ Article* is hard to understand at a glance. However, if the user realizes that this meta-path talks about the *Cannes Film Festival* then he can replace that meta-path with this more meaningful label.

hPathSim allows users to input a meta-path YAML file into the system. They can define meta-paths flexibly and give a label to it. An example, YAML file would look like this:

```
Cannes Film Festival:
    - person: Coen brothers
    - *
    - location: France
```

Would replace any paths within a meta-path of the format *person: Coen Brothers ⇝ * ⇝ Location: France* with *Cannes Film Festival*.

3 Implementation

When hPathSim first runs, it first reads user-specified files that describe the nodes and the links in the heterogeneous information network. Optionally, the user could also specify another set of files that describe the hierarchy of types in the network so that users could perform drill-down or roll-up OLAP operations on the network.

Then, hPathSim builds the graph and Hierarchy Forest, as discussed in Section 2.2.2. After these two data structures have been built, the user will be presented with an interactive console. There are 7 key operations that the user could execute in the console:

- *similarity(node1, node2)*: Executes similarity search between node1 and node2.
- *drill-down(categoryName, categoryValue)*: Executes drill-down OLAP operation on the dataset on user specified hierarchy name and value.
- *roll-up(categoryName, categoryValue)*: Executes roll-up OLAP operation on the dataset on user specified hierarchy name and value.
- *add-constraint(type, id)*: Add a constraint on the meta paths used in the similarity measures as discussed in Section 2.2.3.
- *delete-constraint(type, id)*: Delete a constraint that the user previously added.
- *print-meta-paths(node1, node2)*: Display the meta-paths used to compute the similarity score between node1 and node2.
- *print-network-statistics()*: Display the network statistics (i.e., degree distribution, clustering coefficient and average path length) of the graph that the user is currently working with.

4 Evaluation

In this section, we begin with describing the datasets used in our experiments before evaluating the performance of HPATHSIM on those datasets.

4.1 Datasets

We test HPATHSIM on two different datasets.

The DBLP Hierarchical Dataset is similar to the previous one. Except, that now the relations are as follows: Paper↔Area, Author↔Area, Conf↔Area, Term↔Area. Furthermore, Area and Conf entities form a hierarchy. This dataset is much more smaller than the previous one, only 7.9MB. There are 70536 vertices in this dataset with each node having an average degree of 9.0 with a standard deviation of 74.8. The average path length is 4.0 with a standard deviation of 1.10.

Finally, the last dataset is from the New York Times. It represents a comprehensive dataset that is both heteregenous and also hierarchical. The network is 650MB, with entities: Article, Location, Organization, Person, Topic. Furthermore, each one of these entites except Article has its own hierarchy. There are 181874 vertices in this dataset with each node having an average degree of 10.0 and a standard deviation of 73.7. The average path length is 4 with a standard deviation of 0.97.

Graph	Average (s)	Stddev
DBLP Full	5.26	0.042
NYT Full	5.12	0.157
DBLP DB	1.41	0.03
NYT Jordan	1.70	0.04

Table 1: Average time taken and standard deviation for similarity search on random pairs of entities in the DBLP and NYTimes datasets.

4.2 Experiments

There are three main metrics that we would like to evaluate the performance of our HPATHSIM framework on: the runtimes similarity search query, drill-down and roll-up OLAP queries. Since the underlying similarity search algorithm used is the same as PathSim, the accuracy of HPATHSIM is also the same and thus, we do not display the comparison.

For the similarity search, we measure the times it takes to execute similarity search on a pair of nodes with meta-path length of 4. The evaluation was repeated 5 times. Table 1 shows the time taken on 4 graphs. The first two are the full DBLP dataset and the full NYTimes dataset while the last two are a subgraph of the DBLP dataset whose vertices' "area" dimension is database and a subgraph of the NYT datasets whose vertices' "country" dimension is Jordan. We can see that although, we increase the graph size considerably with NYT Full our runtimes are about the same. We can also see that the standard deviation for similarity search is low, hence our results should be stable.

For the OLAP operations, we measure the time it takes to perform drill-down and roll-up under specific categories. For the DBLP data the drilled-down sub-

Graph	Average (s)	Stddev
DBLP drill down	0.055	0.024
DBLP roll up	0.0058	0.00093
NYT drill down	0.52	0.06
NYT roll up	19.22	0.91

Table 2: Average time taken and standard deviation to perform drill-down and roll-up on the DBLP and NYT datasets.

graph is four times smaller than the original graph. On the other hand, there was no subgraph in NYT that would allow us to get a network of size four times smaller. So we resorted to a network that was only 10 % smaller than the original one. Table 1 shows the time taken for OLAP operations on both the DBLP and NYTimes dataset. We can see that the drill-down operations are really performant, however roll-up can take a long amount of time when we are dealing with a very large network.

5 Conclusion

In this paper, we built hPathSim, a similarity search framework, that improves on the PathSim. Firstly, even with the pruning algorithm that they introduce in the paper, the original PathSim framework still requires expensive matrix computations that are not feasible in an online setting. We introduce a better storage mechanism, MetaPath-Table, that allows faster materialization of the meta paths to support real-time user queries. Secondly, we built our own tree data structure, Hierarchy Forest, to allow us to perform efficient OLAP operations on the dataset. Thirdly, we also allow users to specify constraints on the meta paths that are used in the similarity measures.

5.1 Future Work

For future work, we plan to implement better search algorithm than our current implementation. A meta-heuristic optimization technique such as ant-colony optimization algorithm [1] will be better at navigating the graph with the constraints that the user added.

Another area of future will be to make our framework more scalable by running it in a distributed fashion. There have been previous works that implement distributed graph processing framework such as GraphLab [3] and Graphx [6].

5.2 Open Source

hPathSim is open source and is available for download at `https://github.com/hilfialkaff/newsnet`.

References

[1] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[2] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.

[3] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*, 2010.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[5] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB11*, 2011.

[6] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

[7] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. ACM, 2007.