"All roads lead to Philosophy."

Web query classification using analysis of inter-article relationships in Wikipedia

Ashwin Sethu Baskaran
Dept. of CISE, UF
ashwin421@ufl.edu
UFID: 81542347

Pranav Ravichandran
Dept. of CISE, UF
pranavrc@ufl.edu
UFID: 19498175

Man Mohan Devineni
Dept. of CISE, UF
manmdevineni@ufl.edu
UFID: 36069032

Suhas Tumkur Chandrashekhara
Dept. of CISE, UF
schandrashekhara@ufl.edu
UFID: 49497535

ABSTRACT

Wikipedia has an internal graph structure based on the links between articles. The project aims to explore and exploit this structure between Wikipedia articles for web query classification to give user profile based search results. This idea can be used for developing web query classification on other structured databases like Wikipedia.

Our approach is that we use the user search history to develop a user profile using tagging. This user profile will help us in having a user based query classification. First we will have a view on the structure of the Wikipedia Articles and how they are related to each other giving us an overview on the network topology. Then we describe the importance of dividing the user search history into sessions and tag them. Using these session tags we build a weighted graph of tags with weights associated to nodes as well. Then we will run a community detection algorithm on this graph to cluster the tags which are close to each other in the actual Wikipedia structure and change the weights accordingly. Finally, we will view how these weighted edges and nodes weights help us in query classification.

Keywords and General Terms

Tagging system, Sessions, Query Classification, Community detection, Wikipedia

INTRODUCTION

On May 26, 2008, Wikipedia user Mark J published an article about an uncanny discovery he had made on the site. Quoting Wikipedia:

"Clicking on the first link in the main text of a Wikipedia article, and then repeating the process for subsequent articles, usually eventually gets one to the Philosophy article."

Though Wikipedia was not consciously designed this way, this pattern of ancestral relationships developed out of the intrinsic nature of the first statement in a Wikipedia article. By convention, it starts with a categorization or overarching topic for the subject article. The significance of the first link is that it usually takes us to the broader subject or topic that is under question. If we continue doing this, we traverse up a graph of domains that get more and more abstract until we get to Philosophy, the all-encompassing field of study.

Bacon leads to Philosophy. Here's an example of a road that leads to Philosophy from what is a seemingly unrelated and unassuming article:

Bacon → Pig → Even-toed ungulate → Mammal → Clade → Organism → Biology → Natural Science → Science → Knowledge → Awareness → Consciousness → Quality → Philosophy

Here's another example, starting at National Collegiate Athletic Association:

National Collegiate Athletic Association → Non-profit association → Organization → Entity → Existence → Ontology → Philosophy

It's clear that there exists an implicit graph of relationships in Wikipedia based on fields, subfields and topics.

There has always been a problem of assigning a 'category' to an ambiguous search query. A user searching for the term Apple could be a farmer, a software engineer, or a stockbroker. Web query classification can help the user get better, more personalized results for his search queries. Most of the current web query classification systems depend on a query-enrichment heuristic, like retrieving the top most documents from the search results of each query. Some systems may also use an external document classifier to identify a 'category'. The goal of this project is to use the inherent structure in Wikipedia to build a session based tagging system, which could aid in web query classification.

For instance, a programmer searching for Java may not really be pleased with results of the island Java all over his search engine. Wikipedia article traversal has two different routes for the two different cases. Unlike search engines, let's use the road to Philosophy,

Java → Programming Language → Formal language → Mathematics →Quantity → Property → Modern Philosophy → Philosophy

Java → Island → Continent → Land → Earth → Planet → Astronomical Object → Entity → Existence → Ontology → Philosophy

However, we cannot build an effective web query classification system in a case where a new user has just started his session and made only one or two queries. Our system is intended to incrementally strengthen the user's profile as the number of searches keep increasing.

We take predetermined periods of time called sessions and tag each of them with the parent topic that is most common to most of the topics in the session. For instance, a session with queries apple, mango, tomato and orange, will be marked with the tag 'Fruits'. We then make use of the community structure and the relationships between the tags to come up with an efficient web query classification system.

As the user searches more and more, his search behavior and context will be evident from the distribution of his tags. This can be used to suggest related articles, personalized for him.

RELATED WORK

Community Detection

The aim is to identify groups of entities which are similar to each other, in terms of specific attributes. There exist a number of algorithms which facilitate community detection for large, complex networks. The Blondel algorithm is an apt choice for this purpose. However, Girvan – Newman algorithm (named after Michelle Girvan and Mark Newman) works well on weighted graphs

The Blondel algorithm is a greedy optimization method which attempts to optimize the modularity. The value of modularity is defined within a range between -1 and 1. It compares the density of links between nodes in a community to the density of the links between different communities.

The Blondel algorithm has been used with success for multiple networks of different types and for sizes up to hundred million nodes and billions of links. It can even identify a hierarchy of communities by looking for sub-communities within communities.

The algorithm is divided into 2 phases that are iterated repeatedly [4]. The first phase involves assigning a community to every node. This would result in as many communities as there are nodes. Then, consider placing a node from one community to another and observe the modularity gain. The node is then shifted to the other community for which modularity gain is maximum. But, this shift

occurs only if the gain is positive. Else, the node stays in its original community.

The second phase involves building a new network whose nodes are the communities identified in the first phase. The weight of the links between any two nodes would be the sum of the weight of the links between the nodes of these two communities. At the completion of the second phase, the first phase is applied again to this new weighted network and this process occurs iteratively. This ensures that a hierarchy of communities is unveiled and would be one of the best ways while handling large and complex networks.

This simple approach offers many advantages. Firstly, the steps are intuitive. Secondly, this algorithm is extremely fast. This is primarily because the number of communities decrease by a large extent, just after a few passes, thus limiting the complexity only in the first few levels. Computing the modularity gain is simple too. Although the algorithm seems to run in O(n log n), the exact modularity optimization is a NP-hard problem [5].

The Girvan–Newman algorithm *[7]* *[8]* is a hierarchical method (this technique arranges the network into a hierarchy of groups according to a specified weight function) used to detect communities in complex systems. It detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities. Instead of trying to construct a measure that tells us which edges are the most central (most important/highly connected) to communities, the Girvan–Newman algorithm focuses on edges that are most likely "between" communities.

This algorithm uses the “Edge Betweenness” centrality as a weight function. The "Edge Betweenness" of an edge is the number of shortest paths between pairs of nodes that run along it. If a network contains communities or groups that are only loosely connected by a few inter-group edges, then all shortest paths between these different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness (at least one of them). By removing these edges, the groups are separated from one another and so the underlying community structure of the network is revealed.

The algorithm is divided into 4 steps [8]:

1. The betweenness of all existing edges in the network is calculated first.
2. The edge with the highest betweenness is removed.
3. The betweenness of all edges affected by the removal is recalculated.
4. Steps 2 and 3 are repeated until the desired communities are detected (A threshold on the Edge Betweenness value can be used as a stopping criterion).

The fact that the only edge betweennesses being recalculated are only the ones which are affected by the removal, lessens the running time of the process. However, the betweenness centrality must be recalculated with each step because the network adapts itself to the new conditions set after the edge removal.

Query classification

The paper Improving Automatic Query via Semi-supervised [1] Learning which was published in 2005, describes how to classify the user queries with the combination of manual matching and supervised learning on search log data. Here, they used combination of techniques of exact matching and Supervised Machine Learning (Margins Algorithm) to develop approach of selectional preferences in which a weighted combination of evidences is taken into consideration. The main drawback of this approach is the amount of human effort that it takes.

As an improvement on above mentioned approach, a paper was published in 2009, understanding user’s Query intent with Wikipedia [2] which deals with three major issues in query intent classification

- Intent representation
- Domain Coverage
- Semantic interpretation

Here Wikipedia concepts are used as intent representation and through browsing the category for its siblings and nodes it is linked to, we can get a huge set of information. The intent query is propagated through the structure using Markov Random walk algorithm. Each query is mapped to the Wikipedia concept and the intent is found by examining intent probability resulted from the intent propagation. Explicit Semantic Analysis (ESA) [3] is used if proper match is not found.

The paper Social Network Analysis of Web Search Engine Query Logs [6] states that Search engine query logs are a good resource that records users' search histories and thus the necessary information for such Web query classification studies. Query logs capture explicit descriptions of users' information needs. Logs of interactions that follow a user's query capture derivative traces that further characterize the user and their interests which helps in building the user profile.

The paper proceeds by building an undirected of queries. The graph is constructed by modeling query patterns existing in users' query sessions, based on real-world Web search engine query log data. In these query networks, distinct queries are represented by the nodes of the graph while semantic relatedness amongst these web queries are represented by the edges of the graph. The paper also suggests using statistical measures, i.e. collocation, weighted dependence and mutual information, to capture the strength of such semantic relatedness. It makes use of Newman clustering to extract community structures from query networks, which are potentially capable of representing users' underlying search interests in their search processes

Though we can find existing work on Community Detection and Query Classification, we aim to create an integrated system of tag-based web query classification, where we suggest search results to a Wikipedia user, based on his prior article queries.

QUERIES, SESSIONS, AND USER PROFILE

It is very intuitive that we need to build a user profile making use of the history of queries of that particular user, in order to achieve user based search results. One way to keep track of the queries of a particular user, is to construct a graph with the queries as the vertices and the connectedness between the search queries as the edges and keep extending the graph every time the user searches for a new search term.

Graph construction can be achieved by connecting every query node to every other query node with an edge. We can then assign edge weights based on the number of common ancestors in their respective roads to philosophy. From this assignment of edge weights it follows that the more the edge weight the two query nodes are semantically more closely related. We could then run a clustering or a community-detection algorithm on the graph using Girvan-Newman modularity clustering [7] [8] which works on weighted graphs to group query nodes into category-based communities.

If we follow this approach we will essentially be connecting each query node to every other query node. However, if the user's query network has a million nodes, we will end up constructing a very complex complete graph. Also, this construction of the graph would be time expensive. We cannot possibly connect a million nodes to each other and compute all their edge weights.

To overcome the above mentioned issue, we came up with a novel approach of segmenting the user query history into tagged sessions. We make a logical assumption here, that in a short, designated time-span, a user is very likely to visit related articles and content rather than making unrelated haphazard search queries. This assumption makes sense because a user who is trying to find out more about the latest updates on the satellites orbiting the Earth, would in most likelihood would search a bunch of topics related to satellites in a particular time span rather than deviating from the topic. Sessions are such time periods within which all visited topics are assumed to be connected. The

exact length of the session can be empirically determined.

We now consider every session individually and tag each one of them with the parent topic that is most shared between the topics in the session. For instance, a session with queries apple, mango, tomato and orange will be tagged with the topic Fruit.

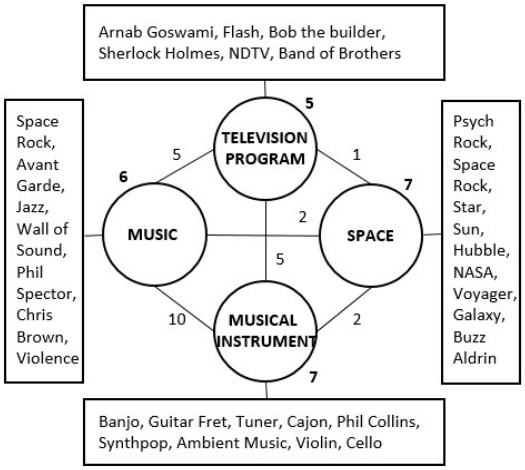

Fig. 1: A graph depicting the clustering and edge weights and node weights

In Fig. 1, all the entries within the rectangles are the search queries entered in a single session. We can observe from the Fig. 1 that all the search queries in a rectangle have been grouped into one set and have been tagged with the parent topic that is most shared between the topics in the session.

Once we have tagged all these sessions, we build a graph with these tags as nodes. We assign node weights based on the number of related queries (a totally unrelated search query in a session will not be considered) in the associated session. For instance, in Fig. 1, the tag MUSIC is assigned a node weight of 6. We can see that there were seven search queries in that session in total, but, only six of them were related to music. Thus the node weight six is assigned to the tag MUSIC.

A tagged node is then connected to every other tagged node with an edge whose weight is the number of common ancestors the two tagged nodes share. From Fig. 1, we can see that the two tagged nodes MUSIC and MUSICAL INSTRUMENT are connected with an edge whose edge weight is 10. This indicates that they are more related to each other and that they share a higher number of common ancestors. However, we observe that the tagged nodes MUSIC and SPACE are connected to each other with an edge with weight as 2. This indicates that they are not so closely related to each other and that they do not share a higher number of common ancestors.

After such a graph with all the tagged nodes are built, the tagged nodes are connected to each other and assigned node weights, and after the edge weights have also been assigned to edges connecting the tagged nodes, there may be a high possibility that a user searched for MUSIC related topics in two different sessions. In such a scenario, we would ideally want the two separate MUSIC sessions to be grouped together into a single node and the combined node to have a cumulative node weight.

So, in order to achieve the above mentioned tag clustering we use community detection. We use the Newman's modularity clustering algorithm which helps in community detection of weighted graphs. From Fig.1, the two tagged nodes MUSIC and MUSICAL INSTRUMENT would likely be clustered together to form a single community and would be assigned a cumulative node weight of 13 (sum of node weight of MUSIC: 6 and MUSICAL INSTRUMENT: 7).

Once we have completed the clustering and the user profile is generated for a particular user, we can make use of this graph to resolve the ambiguity of a new search query that the user may enter in a new session. In such a case, the search results are ordered based on the node weights of the individual tagged communities. For example, when a user searches for *'Space Rock'*, two tagged communities MUSIC and SPACE exist in the graph which include the search query. Based on the node weights of these two tagged communities, the search results will be ordered. In this particular case, as MUSIC has a higher node weight of 13 as opposed to that of SPACE whose node weight is 7, the search results will be ordered as '*Space Rock' (MUSIC) followed by 'Space Rock'* (SPACE).

IMPLEMENTATION

The implementation so far has been to script the process of starting from an article or a node, and visiting all the first links recursively, until we reach the article Philosophy. The code for the implementation is explained below:

Parsing the article's markup

We have a parent class ExtractLink, which consists of routines that help to parse a HTML string and return the first valid link based on some conditions. Here are the routines:

__init__() We use the parsing library BeautifulSoup to convert the HTML of the article into the parser's native format, and then we extract the tag with the ID mw-content-text, as this is the tag that contains the main content of the Wikipedia article.

get_paragraphs() This method finds all the tags that are either a <p> tag or an <ul> tag among the top-level descendants of our parser's output. We do not recurse beyond the top-level as this is redundant and our target links will be present in top-level paragraphs only.

is_valid_tag() A boolean function that returns True if a tag is either a paragraph <p> or an unordered list <ul>, and False if not.

iterate_links() This method takes the output of *get_paragraphs(),* which is a list of all the valid paragraphs in our HTML. It iterates through each paragraph, and for each paragraph, it iterates through each link, checking if the link could be a valid link that could be returned. It uses the *is_valid_link()* function to check the validity of these links.

is_valid_link() Internal links to other articles in Wikipedia are of the format */wiki/article_title*. A link can be valid only under the following conditions:

1. It is not a link to an external webpage.
2. It is not an anchor link (Hashtag that links to another section on the same page).
3. It resembles an internal article link. That is, it cannot contain links to internal Help articles like */wiki/Help:article_name*. So it cannot contain a colon.
4. It cannot be inside parentheses.

This function returns True only if none of the above conditions are met.

get_first_link() This acts as a wrapper function to call the other routines in logical order to return a dictionary with the values of *next_link, next_title* and text. This contains information about the first link of the article that we can traverse to.

Iterating through the articles up to Philosophy

We have another parent class named *IterateArticles* that consists of the functionality to traverse from the start article up to the article Philosophy.

__init__() We initialize the *start_page* and *base_url* properties for use inside the class. The *base_url* is the url that Wikipedia provides for searching. The url is https://en.wikipedia.org/w/index.php?search=

get_page() This function takes a web address and fetches the HTML content from that address. In our case, we use it to extract HTML from Wikipedia articles.

traverse() This is the main function of the class that continuously fetches HTML of every article, extracts the first link, finds the next article to go to, and repeats the process until it gets to *Philosophy*. It appends all the traversed articles into a list and returns that list. If at any point in the traversal, it finds an article that has already been visited, it detects a loop and exits the program.

Segmenting query sessions

The *segment_sessions()* function segments query sessions based on a session interval. Since we assume that searches in a single query session are related to each other, we divide the query list into sessions, each spanning a certain interval.

Associating a single session with a tag

The *TopicRelation* class contains subroutines to build article paths of articles in a single session to the node Philosophy, and then associate them with a tag. The tag is determined based on the most common ancestor of all the articles that holds the most weight.

__init__() We initialize the queries property of the class object as a list of sessions using *segment_sessions().* For every article in each session, we run *build_paths()* to create a dictionary of article to article path objects. Then for each article path in *article_paths*, we find the most common ancestor among their paths. For instance, if five articles go through Music and two go through Space, Music would be the tag of the session.

build_paths() Subroutine to build article paths to philosophy from a list of articles. Returns a dictionary of article - article path relationships.

most_common_ancestor() From a list of article paths, picks the most common ancestor traversed. The higher up a node is, the lower significance it has (Philosophy, for instance, is covered by all paths but cannot be the most common ancestor), so we accumulate previous values with 3.0 / (node level) where 3.0 is an arbitrary number to take into account the significance of the depth.

Building and clustering a graph of topics

The *ClusterTopics* class contains subroutines to build a complete graph from tags, assign edge weights based on number of common ancestors between two tags, and then run the Girvan Newman clustering algorithm on the graph.

__init__() We initialize the topics property of the class object, which is the output of the *most_common_ancestors()* routine from the *TopicRelation* class. Using *make_complete_graph(),* we create a completely inter-connected graph where each node is a topic and each topic is connected to other topics. After we have the graph, we can assign edge weights based on length of common nodes in article paths using *assign_edge_weights()* and then we run Girvan-Newman clustering on the graph using *cluster().*

make_complete_graph() From a list of topics, create a full graph where topics are vertices and each vertex is connected to every other vertex. Initialize default unit edge weights to all the edges.

assign_edge_weights() Between each topic x and y, the function assigns an edge weight which indicates the number of ancestors their article paths share. The function assigns edge weights this way, to all the edges in the graph.

cluster() Runs Girvan-Newman clustering algorithm on the weighted complete graph of topics.

The main workflow

```
 >>> results =
IterateArticles(“Bacon”)
 >>> results.traverse()
bacon -> Domestic_pig -> Even-
toed_ungulate -> Ungulate -> Clade ->
Organism -> Biology ->
Natural_science -> Science ->
Knowledge -> Awareness -> Conscious -
> Quality_(philosophy) -> Philosophy
```

We create an instance of the class *IterateArticles* with the article name Bacon, and then we call the method *traverse()* of that instance, which prints the path from Bacon to Philosophy.

The main file, *main.py*, consists of a sample query log, and code to build the graph and cluster the tags.

```
import cluster

if __name__ == "__main__":
      queries = [('Jupiter', 0),
      ('Saturn', 1),
      ('Mercury', 2),
      ('Venus', 3),
      ('Asteroid', 4),
      ('Cricket', 100),
      ('Table Tennis', 101),
      ('Rugby', 102),
      ('Football', 103),
      ('Chennai', 200),
```

```
        ('Florida', 201),
        ('France', 202),
        ('Australia', 203),
        ('California', 204),
        ('Batman', 300),
        ('The Flash', 301),
        ('Superman', 302),
        ('Jessica Jones', 303)]

        tr =
cluster.TopicRelation(queries, 99)
        ct =
cluster.ClusterTopics(tr.topic_weight
s.keys())
        print ct.cluster
```

Running the code:

To run the Python script from a terminal, we use the following command:

```
python tagspace.py <article-name>
```

For instance, for the articles *bacon* and *Networks*:

```
$ python tagspace.py bacon
bacon -> Domestic_pig -> Even-
toed_ungulate -> Ungulate -> Clade ->
Organism -> Biology ->
Natural_science -> Science ->
Knowledge -> Awareness -> Conscious -
> Quality_(philosophy) -> Philosophy

$ python tagspace.py networks
networks -> Artificial_neural_network
-> Machine_learning ->
Computer_science -> Science ->
Knowledge -> Awareness -> Conscious -
> Quality_(philosophy) -> Philosophy

$ python main.py
```

Returns the final graph of tags, clustered into a dendrogram model.

Once we've acquired the modular graph after running Girvan-Newman algorithm on it, we have a set of clusters, tagged and weighted. For instance, we may have *Planets*, *Music*, *Television Show* and *Musical Instrument* as the tags, of which *Music* and *Musical Instrument* may be clustered together.

The next time the user searches for an ambiguous query, say *Pluto,* we take all the possible articles, say *Pluto the Pup*, a Walt Disney comic character, and *Pluto*, the Planet.

If *Planets* is the most weighted cluster in his graph, *Pluto* the planet is closest in terms of ancestors to the tag *Planets* and *Pluto the Pup* is closest to the tag *Television Shows*.

This results in an ordering of *Pluto* the planet first, followed by *Pluto the Pup*, the Walt Disney character. If *Television Shows* is more weighted than the *Planets* cluster, the ordering of the results is the other way round.

The complete code listing can be found at https://github.com/pranavrc/tagspace

RESULTS

The web query classification built using the inherent structure of Wikipedia has proved to be effective and accurate. Assuming that a graph has already been built based on a few user sessions (Fig. 1), let us consider the case where the user now starts a new session. There are three different scenarios possible now.

Scenario 1:

Let us assume that the user now searches for a term, not belonging to any of the clusters in the graph, say 'apple'. First, the Wikipedia traversals to 'Philosophy' are done, to look for any matching cluster in the way.

Apple → Deciduous → Tree → Botany → Plant → Multicellular → Organism → Biology → Natural Science → Science → Knowledge → Awareness → Consciousness → Quality → Philosophy

Apple → Multinational → Corporation → Legal Personality → Legal Capacity → Natural → Jurisprudence → Study → Learning → Knowledge → Awareness → Consciousness → Quality → Philosophy

But, since the query does not really belong to any of the existing four clusters, the search engine would simply return the default search result for

'apple'. However, this query also gets mapped to the graph and depending upon the other queries that follow in the session, it may form a new cluster.

Scenario 2:

Now, let us take a case where the user searches for a term, which clearly belongs to one of the existing clusters. Suppose the user searches for 'Jupiter', initially, the Wikipedia traversal to 'Philosophy' is done. Along one of the paths, we now find a 'category' which matches one of our existing clusters - SPACE.

Jupiter → Planet → Astronomical Object → Physical Entity → Existence → Ontology → Philosophy

Jupiter → Palm Beach County, Florida → County → United States → Federal Republic → Federation → Union → State → Political Division → Region → Geography → Science → Knowledge → Awareness → Consciousness → Quality → Philosophy

Now, the web query classification system uses this knowledge of the user's history of queries to identify that this user might be more interested in Jupiter (planet) than Jupiter (town). Thus, maintaining the user profile helps us achieve more personalized results for the user queries.

Scenario 3:

Finally, let us consider a situation where a user searches for a query, which might match with more than one of the clusters formed. In such a case, a decision has to be made, as to which cluster this query belongs. A proper decision can be made, considering the node weight associated with each cluster. Suppose the user searches for 'Space Rock'. The first step, traversing to 'Philosophy' in Wikipedia is done. There can again be two different paths leading to 'Philosophy',

Space Rock → Rock Music → Popular Music → Art Music → Umbrella Term → Interval → Mathematics → Quantity → Property → Modern Philosophy → Philosophy

Asteroids (Space Rock) → Minor Planets → Astronomical Objects → Physical Entity → Existence → Ontology → Philosophy

It is identified from the paths that this term may belong to two different clusters, any of which may be true - MUSIC or SPACE. The node weight of each cluster is then considered. A cluster with a higher node weight would correspond to a 'category' that is more frequently searched by the user. Thus, assigning more relevance to the cluster with the higher node weight, it should return a result from the category SPACE than MUSIC. This is because SPACE has a higher node weight and thus, the user is more likely to search for something related to SPACE than MUSIC.

CONCLUSION

We study the Web query classification problem which aims to order the search results better (a personalized search result) for a particular user based on his search history. We propose a novel approach of Tag clustering and community detection which makes use of the inherent structure of Wikipedia to address this issue for Wikipedia articles. Also, our solution performs better with the increase in the size of the user search history.

However, for third-party search engines, if we intend to use the underlying structure of Wikipedia, then it would add a very huge overload on the Wikipedia server (if Google starts scraping Wikipedia articles for every search term it gets, it would cause a huge overload on the Wikipedia server). But, this could be handled in an efficient way if we already have the Wikipedia structure already inherited into the search engine. Also, we could extend our solution to search queries on other databases which possess their own unique inherent structure like Wikipedia. Though these issues are interesting, these would be part of our future work.

REFERENCES

[1] *Improving Automatic Query via Semi-supervised - Steven M. Beitzel, Eric C. Jensen, Ophir Frieder, David D. Lewis, Abdur Chowdhury, Aleksander Kołcz.*
ir.cs.georgetown.edu/downloads/beitzels-Classification.pdf

[2] *Understanding user's Query intent with Wikipedia - Jian Hu, Gang Wang, Fred Lochovsky, Jian Tao Sun, Zheng Chen*
www2009.org/proceedings/pdf/p471.pdf

[3] *Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis - Evgeniy Gabrilovich and Shaul Markovitch*
www.aaai.org/Papers/IJCAI/2007/IJCAI07-259.pdf

[4] *Fast unfolding of communities in large networks – Vincent D. Blondel, Jean -Loup Guillaume, Renaud Lambiotte and Etienne Lefebvre*
arxiv.org/pdf/0803.0476.pdf

[5] *The Louvain method for community detection in large networks*
perso.uclouvain.be/vincent.blondel/research/louvain.html

[6] *Social Network Analysis of Web Search Engine Query Logs - Xiaodong Shi*
www-personal.umich.edu/~ladamic/courses/networks/si708w07/projects/qla.pdf

[7] *Detecting community structure in networks – M. E. J. Newman*
www-personal.umich.edu/~mejn/papers/epjb.pdf

[8] Wikipedia articles on "Hierarchical clustering of networks" and "Girvan–Newman algorithm".
en.wikipedia.org/wiki/Hierarchical_clustering_of_networks
en.wikipedia.org/wiki/Girvan-Newman_algorithm