A Comparative Study of Hierarchical Clustering Algorithms for Tagging Systems

Anisa Allahdadi
BIHE, Iran
anisa.allahdadi@bihe.org

Abstract. With the rapid growth of information on the web, the so-called web2.0 services provide users with a simple way of managing a collection of resources. The collaborative nature of social bookmarking systems allows users to annotate their resources easily and explore other people resources in the network. However, data exploration in such large and complex networks is not always easy, due to lack of organizational structure. Recently several clustering approaches have been introduces to improve the data navigation and information exploration in such systems. In this paper we proposed a framework for the purpose of clustering and visualizing tag spaces to enhance data exploration capabilities. We presented two hierarchical clustering methods and applied them to the del.icio.us data set as one of the most popular social bookmarking systems. We finally made a comparative study on these two approaches regarding the clustering accuracy and time performance.

Keywords. web2.0, tags, collaborative, exploration, navigation, clustering, visualizing

1 Introduction

As the amount of information on the web grows rapidly, the issue of data exploration and retrieval becomes more crucial. Bookmarking systems allow users to annotate their resources with some keywords, called tags, which they consider most appropriate for the content. Users can also share both resources and tags with other users in the network. Tagging system normally consists of three main entities which make a tripartite network of a folksonomy [2]: users, resources, and tags. When a user applies a tag to a resource in the system, a tripartite relationship between the user, the resource and the tag is formed. The study of these relationships among the tripartite entities provides valuable knowledge to obtain:

- Semantic of tags will be easier to understand, as a tag can be defined or explained by the other tags in the group.
- Clusters of similar tags can be utilized for tag recommendation, tag ambiguity alleviation, and tag bundles suggestion.
- Clusters of related resources can be quite useful in the time of searching and recommending similar resources to users.
- The best topic for a resource and some Meta information about it can be offered automatically from representatives of tag bundles.
Community of Like-minded users is provided by putting similar users in a group. The users in each group can share their resources, tagging behavior and so many similar experiences with each other.

The collection of user-generated tags is usually presented in the tag cloud, which is a visual display of tags. Tag clouds take the place of tag lists in which attributes of the text such as font size, color or weight are used to represent the importance or frequencies of the tags. Schrammel et al. [15] performed a research to investigate which layout for tag clouds is the most appropriate one for searching purposes. However there are some inherent limitations in tagging context. Most important of all is ambiguity. While different users apply different words to a resource, and it causes difficulty to find out what the user exactly mean by this word. Acronyms and synonyms are other sources of potential ambiguity as well.

A number of articles [12, 13] claimed that data mining techniques such as clustering, which finds groups of related tags, provide a mean to overcome these limitations mentioned above. With the help of clustering the effect of ambiguity can be alleviated. Gemmell et al. [12] expressed that tags can be aggregated in clusters and their ambiguity can be diminished since the ambiguous tags can be covered by the majority of tags in a cluster. Sbodio and Simpson [13] stated that clustering plays an important role in identifying the concepts of folksonomies and it might also help in automating the process of vocabularies extraction from unstructured folksonomies.

In this work, we focused on investigating the possibility of improving search and data exploration in collaborative tagging systems. We concentrated mainly on clustering like-minded users and similar bookmarks. Further works present that organizing bookmarks according to user interest and tagging manner may lead to even more interesting results. To achieve the mentioned objectives we design and implement a framework in Java called Tag Clustering and Visualizing Framework (TCV). The framework consists of four main sections.

- Data Extraction: allocated for fetching data from del.icio.us tagging system and forming three main matrices of user correlation, bookmark similarity and user similarity.
- Clustering: to cluster obtained data in the previous section based on two hierarchical clustering methods, greedy agglomerative and betweenness divisive.
- Visualization: to demonstrate both original data in the mentioned matrices and clustered data.
- Statistics: for creating excel charts according to the original data and clustering results.

The rest of this paper is organized as follows: Section 2 lists some related works. Section 3 presents the TCV framework and its structure and modules. In Section 4 the experiment results have been reported. And the conclusion and future works have been addressed in Section 5.

2 Related Work

Central to this work is the quality and performance of clustering. Previously in [14] Begelman et al. presented several clustering techniques and provide some results on del.icio.us and RawSugar to prove that clustering can improve the tagging experience. In another direction of work, Gemmell et al. [12] proposed a method to personalize a user’s experience within a folksonomy using clustering. They examined unsupervised clustering methods for extracting commonalities between tags, and use the discovered clusters as intermediaries between a user’s profile and resources in order to connect the result of search to the user’s interests.

Yuruk et al. [22], in a relevant work, proposed a divisive hierarchical clustering algorithm called DHSCAN, that iteratively removes links based on an ascending order of a structural
similarity measure. We did divisive hierarchical clustering in a part of our work as well, but the difference is that we applied betweenness method in order to get more evenly distributed clusters. Newman and Girvan [20], in a similar work to this research, studied betweenness algorithm for divisive hierarchal clustering model, and introduced a measure for evaluating the strength of the communities, called modularity. They applied the algorithm to real networks with known community structure and extracted almost the same structure without much difficulty. Newman [20], however, in a subsequent work proposed a fast algorithm for detecting community structures in networks that was actually an enhancement to the previous algorithm of Newman and Girvan [21] and has a considerable advantageous speed over the earlier algorithm. In our work, we took advantage of modularity formula in both the agglomerative and the divisive clustering algorithms. Our betweenness divisive algorithm is in some manner the implementation of [21] with some distinction for weighted graphs which in practice improved the modularity measured compared to the Newman and Girvan algorithm.

Another similar research is directed by Simpson [16]. He applied two clustering algorithms, ‘tag-co-occurrence divisive’ and ‘betweenness-divisive’ algorithm to two different data sets, and examined the effectiveness and robustness of both algorithms to the different types of data. This research and the similar ones [21], disclosed the poor performance of the betweenness-divisive algorithm for large data sets, as we also confirmed in our experiments (Section 4.1). There has always been a tradeoff between performance and accuracy of an algorithm; this is also true for these two mentioned algorithms. The efficacy and utility of this algorithm is shown in [21] and some other authors [8], [9], [10] utilized this method for their networks as well. Nevertheless, we intend to do more experiments on the accuracy of betweenness-divisive algorithm in future works, as it is rather a remarkable theory.

3 TCV Framework

The proposed TCV framework consists of four main modules: data extraction, clustering, visualization, and statistics.

3.1 Data Extraction

One of the early steps in the process of tag analysis is extracting data from web pages. A number of articles [11], [4], [5], [6] studied record extraction from web pages through identifying a set of segments, each of which represented some data, for example a list of objects. They introduced some methods for this aim, but most of them failed to handle complicated or noisy web page structures. In TCV, we utilized the same concept of data extraction from the HTML tags, but in our specific manner and especially for del.icio.us web pages. We made use of a Java library called htmlparser [3] to parse html tags either in a linear or nested fashion and preserve data in our data structures for later use.

Correlation Graphs. The extracted data are mainly preserved in three correlation graphs or matrices1 which will be explained in the following paragraphs. To construct these matrices efficiently, we made use of the idea of bucket sort algorithm [19]. As bucket sort does not make use of comparison sort methods, its computational complexity is linear and therefore applicable for a large number of data entries.

1 In our design we work on matrices first and then visualize the results in graphs. So the matrices and graphs are applied for the same concept and can be used interchangeably.
User Correlation Matrix. In this matrix the number of matching bookmarks of users-pairs has been counted and reflected.

Bookmark Similarity Matrix. This matrix keeps the compared results of tag sets of bookmark-pairs and calculates the similarity between them. There exists numerous metrics for calculating similarity between two vectors and some of them are listed in Fig. 1.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching</td>
<td>(</td>
</tr>
<tr>
<td>Dice</td>
<td>(\frac{2</td>
</tr>
<tr>
<td>Jaccard</td>
<td>(\frac{</td>
</tr>
<tr>
<td>Overlap</td>
<td>(\frac{</td>
</tr>
<tr>
<td>Cosine</td>
<td>(\frac{</td>
</tr>
</tbody>
</table>

Fig. 1. A set of common similarity measures

We made use of cosine similarity measurement regarding what had been studied in [1] by Xu et al. on accuracy and performance of the above measures.

User Similarity Matrix. This matrix which needs the prepared data in both previous matrices creates user similarities by analyzing the tagging habits of the users. To do this, we need the knowledge of bookmarks’ correlation of user-pairs (user correlation matrix) and the measure of similarity of bookmark-pairs (bookmark similarity matrix).

3.2 Clustering

Greedy Agglomerative Hierarchical Clustering. Agglomerative clustering is a bottom-up clustering approach where each cluster is a subset of a bigger cluster and this trend continues until all the nodes located in at least one cluster. The classic example of agglomerative clustering is species taxonomy.

In the agglomerative clustering approach in TCV, we applied greedy algorithm based on the modularity concepts addressed in [20]. Modularity is a measurement to determine the strength of division of the network to densely connected components with sparse connections between components.

Let \(e_{ij}\) be the fraction of edges that connect the vertices in group \(i\) to vertices in group \(j\). \(e_{ij}\) which defines the edges end in group \(j\) are actually equal to \(e_{ji}\) because of the symmetric characteristics of networks in undirected graphs in this study. \(e_{ii}\) therefore is the fraction of edges in a group connect all the nodes inside the cluster \(i\). To get dense components with sparse connections, increasing the sum of edges in a cluster \(\sum i e_{ii}\) could be an appropriate step to gain higher modularity. However this sum is not precise enough as it could reach to its maximum value when no clustering is done and all the nodes are in the same cluster. Newman suggests calculating the sum \(\sum i e_{ii}\) and subtracting the value that it would take if edged where placed ran-
domly. Here another term is introduced called $a_i$ that is the fraction of edges connected to vertices of group $i$.

$$a_i = \sum_j e_{ij}$$  \hspace{1cm} (1)

Therefore the modularity formula is represented in the following format.

$$Q = \sum_i (e_i - a_i)$$  \hspace{1cm} (2)

In the greedy approach for agglomerative clustering, in each step we select two clusters which merging them maximize the modularity value.

$$\Delta Q = Q_i - Q_j = (e_y - a_y a_j) - (e_y - a_y a_j) = 2(e_y - a_y a_j)$$  \hspace{1cm} (3)

And the pseudo code of the greedy agglomerative algorithm is given in the following piece of code.

ALGORITHM AGGLOMERATIVE($G=\langle V, E \rangle$)
//there are n clusters each contains one node
s:=0;
while (s<STEPS) do{
    CalculateModularityofClusters();  //Eq.2
    q1,q2:=GetTwoBestClusterstoMerge();  // Merging q1 and q2 gives the highest modularity based on Eq.3
    MergeClusters(q1,q2);
    // q1,q2 makes one new cluster
    s:= s+1;
}

Betweenness Divisive Hierarchical Clustering. Hierarchical divisive is a top-down method of clustering which generates clusters by sub-dividing the single cluster containing the entire network at first. At each step the least similar vertices or the most distant ones are selected and the connecting edge is removed until no more edge is remained.

Instead of cutting the least similar edges in the correlation graphs, we intend to remove the edges with the highest betweenness, as it has occasionally studied in the past and it brings well spread clusters based on the modularity definition (Eq. 2). Betweenness is a measure that disfavor edges inside communities and get points to those that exist between communities [21]. The edges with the highest betweenness are allocated to connect many nodes together and therefore many clusters, so by removing them the inherent clusters will appear step by step. The betweenness divisive method consists of two main processes: vertex value and betweenness value calculation.

**Calculate Vertex Values.**

1. The initial vertex $s$ is given, distance $d_s=0$ and weight $w_s=1$.
   - $d$ is the distance from the source node to this node in breadth-first search (BFS).
   - $w$ is the weight of each node shows how many distinct paths there are from the source node to this node.
2. Every vertex $i$ adjacent to $s$ is given, distance $d_i=d_s + 1=1$ and $w_i=w_s=1$.
3. For each vertex $j$ adjacent to vertices $i$ we select one of these cases: a) if $j$ has not been assigned values for $d$ and $w$, $d_j=d_i + 1$ and $w_j=w_i$. b) if $j$ has already been assigned and $d_j=$
\( di+1 \) then \( w_j = wi + wj \). c) If \( j \) has already been assigned and \( dj < di+1 \) the weight is remained unchanged.

- In the b branch a vertex is being considered more than once, meaning that there are some ways from source to this node, so adding the weight value ensures that we consider all the distinct paths in our calculation.

4. Repeat from step 3 until all the vertices have been assigned a distance and weight value.

**Calculate Betweenness Values.**

5. Find every leaf vertex \( t \). A leaf vertex is a vertex that no vertex should pass through it in its path from \( s \).

- To distinguish the leaf vertex we should find vertices for which there is no neighbor with the higher \( d \) compare to \( d \) of the leaf vertex.

\[ d_i < d, \quad i \in \text{neighbors} \]

6. For each vertex \( i \) neighboring \( t \) assign the value to the edge from \( t \) to \( i \) of \( \frac{W_i}{W_j} \).

7. Now we go through other vertices \( j \) which all their below neighbors edges are already calculated. By below neighbors we mean neighbors vertices with lower \( d \). The edge from vertex \( i \) to \( j \), with \( j \) being farther from \( s \) than \( i \), assign the value that is 1 plus the sum of the values on the neighboring edges immediately below it all multiplied by \( \frac{W_i}{W_j} \).

- Here is the modification point in the main algorithm: instead of adding just 1 to the above sum, we add 1 plus the weight of the edge showing that some edges are more likely to be between other nodes according to their weights.

8. Repeat the step 7 until node \( s \) is reached.

When the calculation for a single source is finished, the process will be start over until all nodes become source once; we then have the betweenness values for each edge which is the sum of scores for the different source nodes.

![Figure 2](image.png)

**Fig. 2.** Calculate edges values when ‘a’ is the source node
3.3 Visualization

For visualizing clusters before and after clustering, we make use of **prefuse** visualizing toolkit [17], which is a software framework for creating dynamic visualizations for both structured and unstructured data [7]. We applied various layouts of **prefuse** to demonstrate our original graphs and the communities formed after the clustering process. In Fig. 3 the process of visualizing in the TCV framework is demonstrated.

![Fig. 3. How visual toolkit works in TCV framework](image)

The input graph which is already saved in a text file is given to VisUtility class that creates graphML [29] out of the input graph. VisualToolkit class is able to create different layouts of the output graph when it is supplied with the proper graphML. Some samples of the Bookmark Similarity Graph represented in different layouts are demonstrated in Fig. 4.

![Fig. 4. Different layouts of **prefuse** for bookmark similarity matrix](image)

3.4 Statistics

In the statistic part of the TCV framework, performance of the clustering methods can be compared and demonstrated in different excel charts. The clustering algorithm and the matrix are selected and the chosen type of the chart will be drawn in this step. With the help of a Java chart library, **jfreechart** [18], various excel charts can be drawn to have different looks of the clustering process for all three matrices.
4 Experiments and Results

This section presents some experiments conducted to consider different aspects of a tagging system which leads to some series of results and conclusions. For several experiments we use two sets of data from del.icio.us, a sample of 20 users from the popular bookmarks and another sample of 50 users from the recent page. Table 1 shows the characteristics of two samples of the data set.

| Table 1. Statistics of two samples of data sets |
| User Source | Users No. | Bookmarks per User | Total Bookmark No. | Tag No. |
| Sample 1 | popular | 20 | <=300 | 1714 | 1603 |
| Sample 2 | recent | 50 | <=300 | 3992 | 3832 |

4.1 Agglomerative vs. Divisive Approach

In this section we do a comparative study on agglomerative and divisive clustering methods regarding the modularity and timing issues.

Modularity.

![Fig. 5. Modularity vs. steps No. for agglomerative and divisive method](image)

In Fig. 5 the horizontal axis is the number of steps and the vertical axis is the modularity measured in each step. As the experiment in sample 1 shows, the maximum value for modularity is achieved in the agglomerative algorithm (about 0.42). Although modularity value closer to 1.0 is an indicator for a better clustering, in real networks, however the highest reported value is 0.75. In practice, Newman [20] found that modularity values around 0.3 indicate a strong community structure for the given network. Here the agglomerative method achieves about 7% higher modularity rather than the divisive method.

In sample 2 diagram, the superiority of agglomerative method to the divisive algorithm is even more observable. The maximum modularity value in the divisive approach is 0.37 while it is 0.47 for the agglomerative approach. (about 10% improvement for agglomerative)
Timing. Although the increasing speed of computers will probably alleviate time limits in the coming years, the time complexity is still an important issue in most of the algorithms. Fig. 6 shows a comparison between running time of agglomerative and divisive algorithms in the user correlation matrix of 50 users (Sample 2).

Fig. 6. Measured time in agglomerative and divisive algorithms

The above graph shows that there is a steady upward trend for agglomerative time consumption, while the divisive trend has a sharp increase in the amount of time needed to cluster the same sets of objects. As it is also clarified in [21], the divisive clustering method is entirely computational intensive and it operates in $O(n^3)$ time on sparse graphs. Therefore, for larger graphs with more than 10000 nodes this algorithm is rather inapplicable.

4.2 Betweenness Divisive Method

Fig. 7. Modularity vs. number of clusters for the original formula and the new formula

The new formula which has addressed in section 3.2, step 8 of the betweenness algorithm, deals with the weights of the edges in our similarity graphs. The changes made in this formula in a successful manner increased the overall modularity. As Fig. 7 shows, the new formula obtained better results rather than the original one. For data set of sample 1, the modularity value improved by almost 1%, and as the diagram of sample 2 shows, the maximum modularity gained by the original formula, for un-weighted graphs, is almost 0.37 while it reached a peak at 0.42 for our weighted graph (about 5% improvement).
5 Conclusion and Future Works

In this work, we proposed a framework for clustering and visualizing the data of a well-known bookmarking system, del.icio.us. The main goal was to improve the data exploration and data navigation in collaborative tagging systems. Two hierarchical clustering models have been analysed and implemented, greedy agglomerative and betweenness divisive. We also made some modifications to the main betweenness divisive algorithm and proposed a new formula to improve the accuracy of the mentioned algorithm. However, according to our experiments in section 4.1 and 4.2, we observed higher accuracy and superior performance in agglomerative methods rather than divisive approach. In fact betweenness divisive results proved its inferiority in computational timing and even modularity gained, and therefore it tends to be less widely used rather than the agglomerative methods based on the acquired results.

One of the open issues directly related to our framework is developing a search engine in bookmarking environments and applying clustering to it in order to show improvement in search result over clustered entities. Another further work regarding the divisive clustering algorithms is to improve its speed. Some solutions such as parallelization were proposed in [21], but the issue is still remained open. Another work could be a comparative study of traditional divisive methods and the betweenness divisive algorithm introduced in [21]. Although in [21] the concept of betweenness was addressed as a superior approach, we couldn’t find any research specifically dealing with the superiority of this approach over the traditional one.

Acknowledgement. I would like to thank BIHE\(^2\) for giving me the opportunity to pursue my studies and researches, while it is not possible in Iran for Baha’i\(^3\) scholars. And special thanks to Mr. Farid who is now in prison for the sake of educating Baha’i students! Many thanks for all his supports and devotions. I must thank my supervisors Dr. Hercoc and Dr. Duman for great guidance and advice and my family for all their supports and sacrifices.

References


\(^2\) http://www.bihe.org/
\(^3\) http://www.bahai.org/
17. http://www.prefuse.org/
18. www.jfree.org/jfreechart/