Finding Betweenness in Dense Unweighted Graphs

Brandeis Marshall and Anuya Ghanekar
Computer and information Technology
Purdue University
West Lafayette, IN 47907 USA
{brandeis, aghanek}@purdue.edu

***Abstract*—Social network analysis (SNA) aims to identify and better determine the relationship amongst data in a graph representation.The interpretation of several core SNA measures, *degree*, *closeness* and *betweenness* centrality of a node, have been the subject of extensive research in recent years. We concentrate on the betweenness property, which seeks to determine the relatedness of more than 2 nodes. We propose our *betweenness in unweighted graph* algorithm and compare it to the k-path centrality algorithm on two image collections. By design, our proposed algorithm is less restrictive with the ability to consider any subset of nodes for betweenness. Our findings also show our proposed algorithm has a much shorter execution time as compared to the k-path centrality algorithm.**

***Keywords*-social network analysis; betweenness centrality; social networking graph model.**

I. Introduction

In an effort to deal with the large amount of data being generated in science, research and personally, data models, frameworks and algorithms are designed to reveal information connections that will result in useful knowledge. In many contexts, data remains disjointed with a one-dimensional slice of information. Data duplication and inconsistency are common concerns leading to a lack of confidence in the quality of data. A comprehensive view of information, leading to knowledge, can be obtained using a collection of semi-relevant and overlapping data slices. The ultimate aim is in providing knowledge which allows the end-user to have more confidence in making informed decisions. These measures can be applied in various fields like bioinformatics, image retrieval and supply chain management.

To achieve this goal, social network analysis (SNA) have been used in recent years to find connections amongst data using a graphical representation. Given that search and sort methods are at the center of SNA, the data quality and information flow are assessed through novel implementations of the *degree*, *closeness* and *betweenness* centrality of a node methods, which effectively navigates a social network. The degree algorithm focuses on popularity and frequency of a particular node. The closeness algorithm concentrates on pairwise relationships. The betweenness algorithm considers the relationships amongst more than two nodes.

For this paper, we examine the challenge of applying betweenness in isolating image tags' relatedness, typically used in image search. With the surge of digital photography, image search and retrieval is a complex area of research. Image search and retrieval usually fall into one of two main approaches: content-based image retrieval (CBIR) or image annotation/tagging. At the core, CBIR methods use numerical values while image tagging uses keywords and word phrases to represent an image. However, both methods have their obstacles. CBIR can be time-consuming due to the need for extensive image processing. Image tagging can be error-prone due to lack of reliable verification and validation of manual and (semi-) automated labeling.

According to the prior work on image search and retrieval, image tagging has become the popular choice due to its ease-of-use factor and reduced requirement of computing resources as compared to CBIR approaches. We therefore concentrate on the image tagging approach. We discover that image search and retrieval is a very complex in social networks. Each image is represented as a set of connected tags. When these images are transposed to a graph, we have a multigraph structure. We seek to navigate an image collection's multigraph in order to efficiently determine its betweenness connectivity. High betweenness connectivity depicts that the node has a high impact on the other nodes. We show examples of images and their associated tags in Figure 1. We make the following contributions:

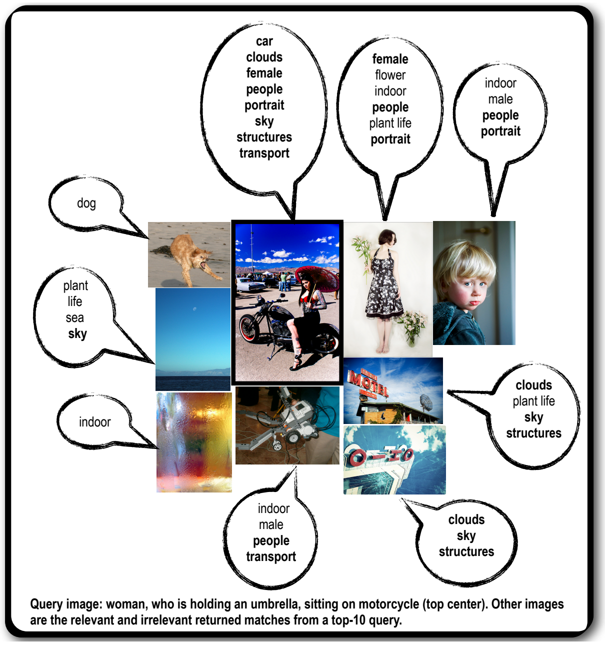
- we present the *betweenness in unweighted graph* algorithm and
- we compare it to the state-of-the-art k-path centrality algorithm on 2 common image datasets, MIRFLICKR and ImageCLEF.

Section 2 discusses the related work for betweenness property in terms of algorithm design, experimental setup and datasets. Section 3 describes the betweenness property and proposes our algorithm for calculating betweenness in unweighted graphs. Section 4 demonstrates the experiments performed on BUG and K-path centrality algorithms. We conclude and summarize the paper in Section 5.

II. Related Work

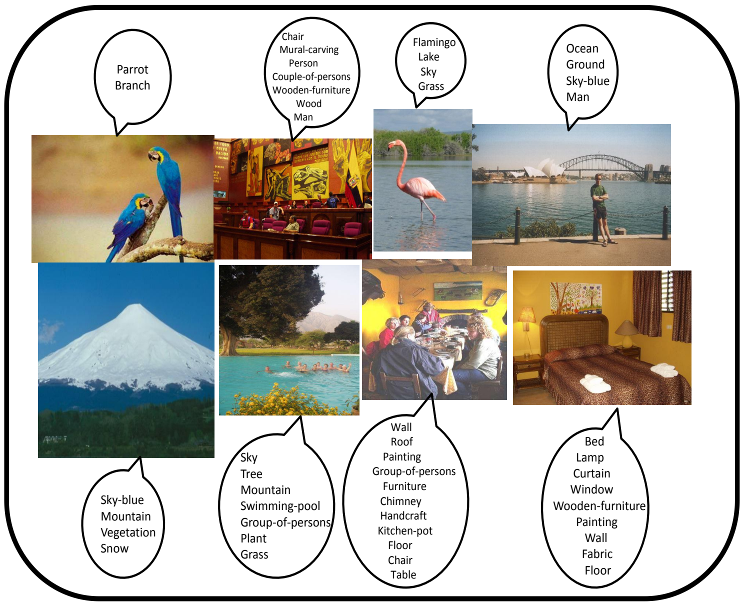
In developing an efficient and effective betweenness algorithm, we must compare prior work [1], [2], [14] with regard to competitive algorithm design and experimental evaluation e.g., experimental setup and datasets.

a) Algorithm Design: Using the K-path centrality algorithm [1], every node selects another node to pass the information to, at random, depending on the weights of the edges and the nodes that have already been visited. It assumes message traversals only along simple paths and of a maximum



Query image: woman, who is holding an umbrella, sitting on motorcycle (top center). Other images are the relevant and irrelevant returned matches from a top-10 query.

(a) Sample MIRFLICKR images with corresponding tags



(b) Sample ImageCLEF images with corresponding tags

Fig. 1: MIRFLICKR and ImageCLEF sample data

length k. The algorithm runs in time $O(k^3 n^{2-2\alpha} \log n)$, where k is the path length, n is the number of nodes. An adaptive sampling algorithm [2] is presented for betweenness centrality. It selects a subset of vertices and performs shortest path computations and then varies the number of samples based on the information obtained from each sample. This algorithm outputs the centrality of all vertices. Okamoto et al. [14] presents an algorithm to determine the top-k vertices by combining the exact and approximate algorithms. Hence, unlike the previous two algorithms, this algorithm is used to determine the k vertices that have the highest centrality, with some error. This algorithm takes $O((k + n^{2/3}. \log^{1/3} n)(n \log n + m))$.

b) Experimental Setup: Alakahoon et al. compare the k-path centrality algorithm with the Brandes [3], RA-Brandes [4] and AS-Brandes [2] algorithms. The experiments show that the k-path centrality algorithm can scale up easily to large networks. The near optimal performance of k-path in both correlation and speedup performance metrics can be achieved when its parameters are set to α = 0.2 and k =ln$(n + m)$, where n and m are the number of nodes and the number of edges in the network respectively.

Bader et al.'s experiments demonstrate that the estimated centrality scores are very close to the exact ones and also reduce the computation time by a factor of nearly 20. They also show that the error variance is within acceptable bounds. The experiments also show the graphs for number of samples/SSSP computations as a fraction of n to depict the amount of work done by the approximation algorithm. Since Okamoto et al. only consider the top-k elements, the computation time is considerably reduced if one is interested only in determining the highly ranked nodes instead of the actual centrality values of all the nodes.

c) Datasets: The scalability of the algorithms are a concern. Bader et al. uses 6 real world graph instances (4 undirected, 2 directed graphs) where the number of vertices varies from 2000 to 9914 and the number of edges varies from 4,435 to 41,601. On the other hand, Alakahoon et al. uses 7 real networks with the number of vertices varying from 2,424 to 82,168 and the number of edges varies from 13,354 to 948,464. Three of these networks use directed graphs while the others use undirected graphs. The edges are unweighted in all the networks except one.

III. Betweenness Property

The betweenness property has its roots in social network analysis theory [8], which considers the use of nodes on a path between a particular source and sink node. A betweenness method is usually costly, especially in complex networks [2], [6]. In addition to betweenness, social network analysis also proposes degree and closeness properties as methods to better understand how data are connected in a graph. In general, data has a semi-structured configuration, including data duplication and naming inconsistency obstacles, led to the need for explicitly and implicitly leveraging relationships within the data. Given the heterogeneous nature of today's data, the challenge is harnessing these data needs across architectures, operating systems and devices.

In prior work, the Social Network Graph (SNG) model [10], [11], [12] aims to integrate a more inclusive data model for incorporating all forms of data from multiple and diverse sources. Currently, SNG has an emphasis on representing personal images and their corresponding annotations. Tagging can be daunting; hence, semi-automated annotation approaches [5], [16] are typically employed. We make use of SNG in identifying a set of attributes, *(who, what, when, how,* *where)*, for an image. Other researchers do not consider this more comprehensive view of an image. For instance, Rattenburg et al. [15] considers "where" the photograph was taken, "what" is occasion at which it was taken and the association between objects of "how" they are related. Golder [7] focuses on images with people and, as a result, infers the picture-taker's social relationships.

SNG's construction consists of a collection of multi graphs, in which each image is represented by a clique connecting the image's tags. At the core of most betweenness algorithm implementations is a shortest path method. The methods differ with regard to graph size, number of edges and, most importantly, navigational techniques. We will describe a state-of-the-art betweenness algorithm proposed by Alakahoon et al., which we will compare to our proposed method in the Experimental Evaluation section.

A. K-Path Centrality Algorithm.

This algorithm attempts to introduce the k-path centrality, which can efficiently compute randomized centrality with accurate results even for a large network. By discovering a way to compute the centrality effectively, the various graphs can be analyzed to determine the effect of one node on another. It aims at reducing the computation time by making two assumptions - consider only simple path (no repetition of nodes) and maximum path length is k (dependent on the network). By using these assumptions, the complexity is greatly reduced as the k-path centrality gives an accurate and quick result for betweenness.

The pseudocode of this method is given as follows: it takes as input a graph G = (V,E), a non-negative weight function on the edges of G, and parameters $\alpha \in$ [-1/2,1/2] and integer κ = f(m,n) where m are the number of edges and n is the number of nodes in the graph. Lines 16-21 compute the sum count[v] that a message originating from all possible source nodes s, goes through v, assuming that message traversal are only along random simple paths of at most κ edges. To compute this, a vertex is chosen randomly such that it has not been visited and an edge exists between the vertex and s, with a probability proportional to the weight of the edge. This sum is used to calculate the centrality in lines 27-29.

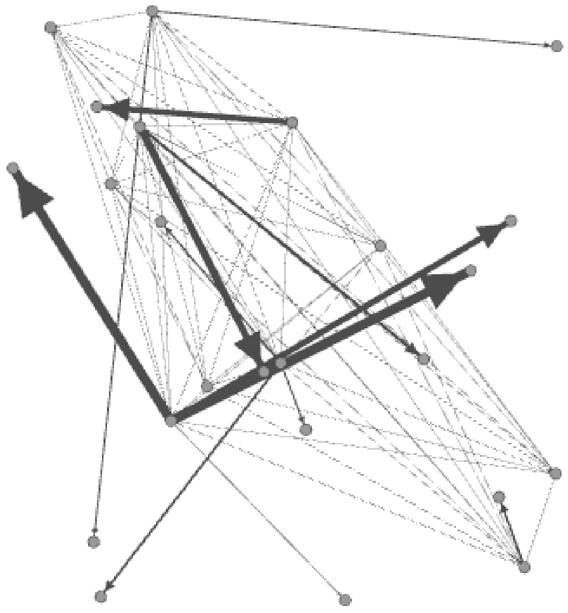
```
1:  function kpathcentrality(graph:$G(V,E)$, array:$W$, int:$k$)
2:  $output$ Array $C_k$ of k-path centrality estimates
3:  N = number of vertices
4:  $\alpha \in$ [-1/2, 1/2]
5:  for $v = 1$ to $N$ do
6:      count[v]=0
7:      explored[v] = false
8:  end
9:  /* S is a stack */
10: $T \leftarrow 2k^2 n^{1-2\alpha}$ ln $n$ ; $S \leftarrow \emptyset$
11: for $i = 1$ to T do
12:     /* simulate message traversal from $s$ containing $l$ edges */
13:     $s \leftarrow$ a vertex chosen uniformly at random from V;
14:     $l \leftarrow$ an integer chosen uniformly at random from [1,k];
15:     explored[s] $\leftarrow$ true; push $s$ to S; j $\leftarrow$ 1;
16:     while $j \leq l$ and $\exists(s,u) \in$ E s.t. !explored[u] do
17:         $v \leftarrow$ a vertex chosen randomly from { $u \mid (s,u) \in E$ and !explored[$u$] } with probability proportional to 1/W (s,v);
18:         explored[v] $\leftarrow$ true; push $v$ to S;
19:         count[v] $\leftarrow$ count[v] + 1;
20:         s $\leftarrow$ v; j $\leftarrow$ j+1;
21:     end
22:     /* reinitialize explored[v] to false */
23:     while S is nonempty do
24:         pop v $\leftarrow$ S; explored[v] $\leftarrow$ false
25:     end
26: end
27: for $v = 1$ to $N$ do
28:     $C_k$[v] $\leftarrow$ kn . (count[v]/T);
29: end
30: return $C_k$;
31: end
```

B. Betweenness in Unweighted Graphs (BUG) Algorithm.

We propose a betweenness method that is best-suited for unweighted dense graphs. We leverage the nearest neighbor relationship SNG nodes and edges. Below we provide our BUG pseudocode that makes execution calls to our closeness method [12]. In our closeness algorithm, we implement Dijkstra's algorithm (beginning at line 8) to find the shortest search path from source node v_i to the target node v_j. Given that our edges are not weighted, we implement a randomized K Nearest neighbor (KNN) to select the neighboring unseen node.

```
1:  function closeness(graph: $SNG$, object: $v_i$, object: $v_j$)
2:  for object $v_y \in SNG$ do
3:      $e(v_i, v_y) = \infty$ // Unknown distance function from source $v_i$ to $v_y$
4:      $prev(v_y) = null$ // Previous node in optimal path from source
5:  $e(v_i, v_i) = 0$
6:  $Q =$ all vertices $V \in SNG$
7:  $v_{curr} = v_i$
8:  while $Q \neq$ null OR reach $v_j$ do
9:      object $v_{close} =$ KNN($e(v_{curr}, v_u)$)
10:     if $e(v_{curr}, v_{close}) = \infty$ then
11:         break
12:     remove $v_{close}$ from $Q$
13:     for each neighbor $v_n$ of $v_{close}$ do
14:         $alt = e(v_{curr}, v_{close}) + e(v_{close}, v_n)$
15:         if $alt < e(v_{close}, v_n)$ then
16:             $prev(v_n) = v_{close}$
17:             $v_{curr} = v_n$
18:             break for-loop
19: return prev
```

Our BUG algorithm first computes the closeness between each node-pair and stores each path in an adjacency matrix (line 3-6). Then, we construct the final path from $v_1, \ldots, v_n$ by identifying when (or if) path overlap occurs (line 9-22). We divide the path overlap into three categories: (1) path 1


(a) MIRFLICKR dataset with 24 nodes and 31,000+ edges

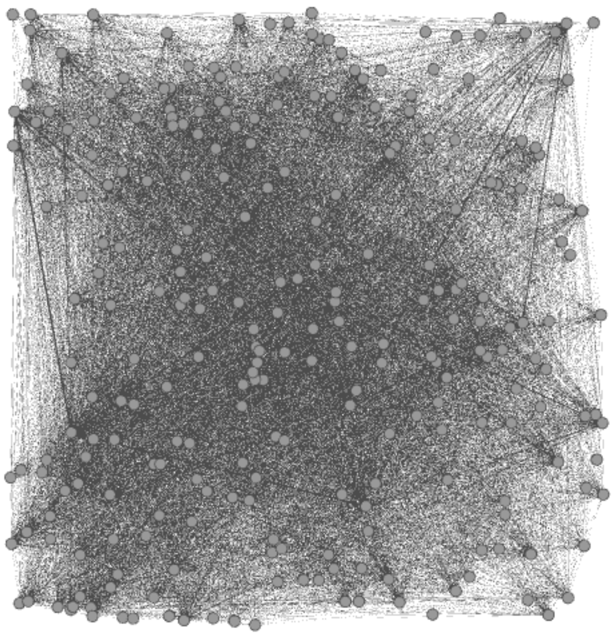

(b) ImageCLEF dataset with 258 nodes and 970,000+ edges

Fig. 2: Visual Representation of MIRFLICKR and ImageCLEF Datasets

and path 2 overlap in path 1 with the node at the beginning of path 2, (2) path 1 and path 2 overlap with path 1 connecting to a substring of path 2, and (3) path 1 and path 2 are disjoint, which results in a union of both paths.

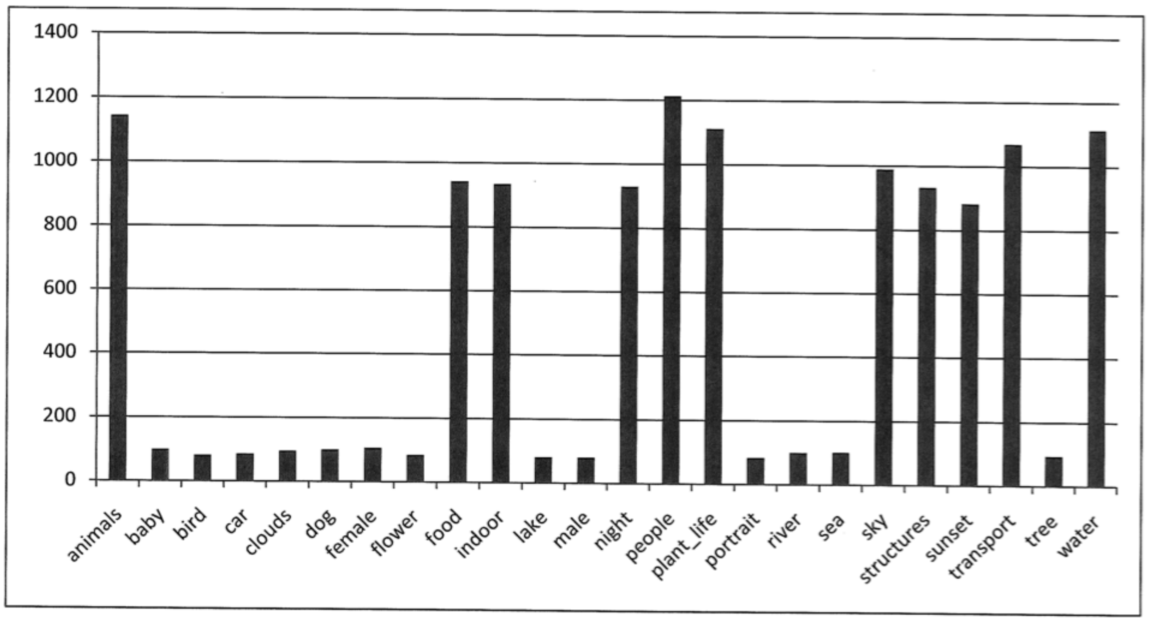
```
1: **function** BUG(graph:$SNG$,objects:$\{v_1, \ldots, v_n\}$)
2: $pmatrix[n][n] = \infty$ // stores the path between each node-pair
3: **for** $i = v_1$ to $v_{n-1}$ **do**
4:     **for** $k = v_2$ to $v_n$ **do**
5:         $path_{ik}$ = closeness($i$ , $k$ , $SNG$)
6:         $pmatrix[i][k]$ = $path_{ik}$
7: $fpath = \infty$
8: $l = v_1$
9: **while** $l \neq v_n$ **do**
10:     $path_l = pmatrix[l][l+1]$
11:     $path_{l+1} = pmatrix[l+1][l+2]$
12:     **for** $j = 1$ to $r$ **do**
13:         **for** $m = 1$ to $s$ **do**
14:             **if** $path_l.get(j)$ = $path_{l+1}.get(m)$ AND $m = 1$ **then**
15:                 $pmatrix[l][l+2]$ = $path_l.substring(1, j)$ $\cup$ $pmatrix[l+1][l+2]$
16:                 boolean flag=true
17:                 break
18:             **else**
19:                 $pmatrix[l][l+2]$ = $pmatrix[l][l+1]$ $\cup$ $path_{l+1}.substring(m, end)$
20:                 boolean flag=true
21:                 break
22:     **if** flag = false **then**
23:         $pmatrix[l][l+2]$ = $path_l$ $\cup$ $path_{l+1}$
24:     $fpath.append(pmatrix[l][l+2])$
25:     l+=2
26: return $fpath$
```

IV. Experimental Evaluation

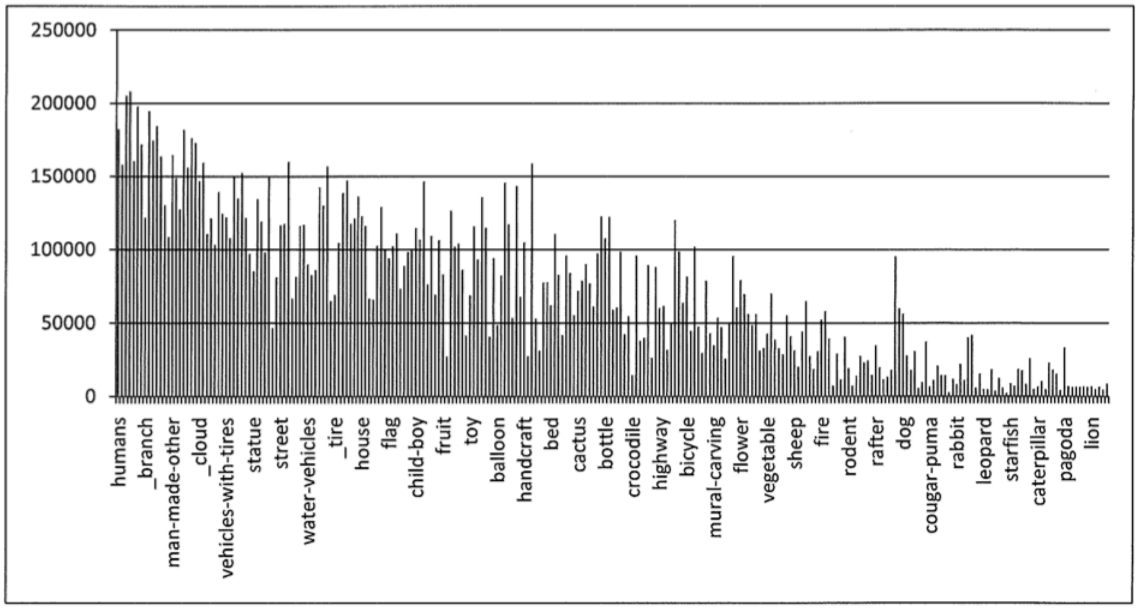
In this section, we describe the experimental results of our BUG methods with that of the k-path centrality method. To perform a more balanced comparison, we test these methods on two datasets: MIRFLICKR-25000 [9] and ImageCLEF Segmented and Annotated IAPR TC-12 [13]. One MIRFLICKR collection consists of 25,000 images downloaded from the social photography site Flickr through its public API. With only 24 annotations, this dataset is very dense having a large number of edges between any two annotations e.g., 'baby' is labeled in 259 images and 'people' is labeled in 10,373 images. MIRFLICKR annotations are arranged in a shallow hierarchical structure of general topic and sub topic categories. The ImageCLEF SAIAPR TC-12 dataset contains segmentation masks and segmented images for 20,000 images and organized in a more complex conceptual hierarchical structure with 258 annotations.

In Figure 2, we display the visual representation of both datasets using Gephi, an open source graph visualization and manipulation software (https://gephi.org). The edge thickness correlates to the frequency of connection between two nodes. We load each dataset into our SNG model, forming a clique of the tags (nodes) for each image. Our MIRFLICKR dataset has only 68 unique edges while the ImageCLEF dataset has 19,509 unique edges.

For our experiments, we execute tests which compare the run times of the k-path centrality and our proposed algorithms on both image datasets. In Figure 3, we display the run time of each tag's betweenness assessment for the MIRFLICKR



(a) MIRFLICKR dataset



(b) ImageCLEF dataset

Fig. 3: K-Path Centrality Results, when K=3

and ImageCLEF collections e.g., x-axis denotes the image tags and y-axis denotes the time in milliseconds. The time on y-axis indicates the average time required to find the shortest path between a node and other pairs of nodes. Each collection's hierarchical structure is revealed by the length of the run time in which image tags located at or near the tree leaves are accessed less frequently. Based on the design of the k-path centrality algorithm, the betweenness evaluation is conducted for each image tag as shown in Figure 3. Our BUG algorithm, on the other hand, first identifies which image tags are of interest and then computes their betweenness. Hence, BUG considers the binomial coefficient in which we find the betweenness of a set of nodes while the k-path centrality focuses in the information-theoretic aspect in which the information flow through a node is assessed. The k-path centrality finds a path only of length k whereas the BUG algorithm finds the shortest path between nodes without any restriction on the path length. The k path algorithm maintains a list of visited nodes and hence requires more overhead. BUG does not require such tracking of these details.

Below we show a path samples from both datasets:

- MIRFLICKR
  - BUG(baby, car, flower) has path baby $\rightarrow$ people $\rightarrow$ transport $\rightarrow$ car $\rightarrow$ plant_life $\rightarrow$ flower and takes 19 ms.
  - BUG(car, lake, tree) has path car $\rightarrow$ transport $\rightarrow$

water $\rightarrow$ lake $\rightarrow$ plant_life $\rightarrow$ tree and takes 7 ms.
  - BUG(clouds, flower, portrait) has path clouds $\rightarrow$ sky $\rightarrow$ plant_life $\rightarrow$ flower $\rightarrow$ people $\rightarrow$ portrait and takes 13 ms.
- ImageCLEF
  - BUG(fabric, cloth, _flock-of-birds) has path fabric $\rightarrow$ cloth $\rightarrow$ humans $\rightarrow$ _flock-of-birds and takes 406 ms.
  - BUG(glass, deer, hedgehog-porcupine) has path glass $\rightarrow$ _ground $\rightarrow$ deer $\rightarrow$ landscape-nature $\rightarrow$ hedgehog-porcupine and takes 558 ms.
  - BUG(diver, beetle, elephant) has path diver $\rightarrow$ beetle $\rightarrow$ animal $\rightarrow$ elephant and takes 415 ms.

These samples showcase the shortest path planning route given each collection's hierarchical structure. The betweenness assessment for the selected three image tags is not a simple and direct path. For instance, the image tags, *baby, car, flower*, are all child nodes within this MIRFLICKR collection with parent nodes *people, transport* and *plant_life*, respectively.

	K-Path Centrality	BUG
MIRFLICKR	519.39 ms	**10.55** ms
ImageCLEF	71678.87 ms	**776.93** ms

TABLE I: Run Time Averages

Table I shows the average execution times for each algorithm-dataset pair. We set k-path centrality parameter $k = 3$ denoting a 3NN information flow. We implement our BUG algorithm with $n = 3$ denoting that we are finding the relatedness of 3 image tags by not setting a restriction on K within the KNN. The BUG algorithm takes a fraction of the run time than that of k-path centrality. In addition, the BUG algorithm generates the path of any number of objects even when the objects do not have a direct edge between them.

V. Conclusion & Future Work

We propose our betweenness algorithm, that is designed for a dense multigraph environment. We performed an experimental evaluation using an MIRFLICKR 25,000 image collection and an ImageCLEF 20,000 image collection, in which we compared our proposed method to the k-path centrality method. Our findings show that the proposed algorithm executed in a fraction of the run time than the k-path centrality method. Additionally, our proposed method is designed with a less restrictive interpretation of betweenness as the shortest path including any subset of nodes is produced. In the future, we plan to work on a weighted closeness and betweenness for larger image datasets, including the MIRFLICKR 1M collection. We would like to better incorporate information theory principles in these methods and assess its impact in a large-scale data environment.

References

[1] T. Alahakoon, R. Tripathi, N. Kourtellis, R. Simha, and A. Iamnitchi. K-path centrality: A new centrality measure in social networks. In *Proceedings of the ACM Workshop on Social Network Systems*, page Article 1, 2011.

[2] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. In *Proceedings of the ACM international conference on Algorithms and models for the web-graph*, pages 124–137, 2007.

[3] U. Brandes. A faster algorithm for betweenness centrality. *The Journal of mathematical sociology*, 25(2):163–177, 2001.

[4] U. Brandes and C. Pich. Centrality estimation in large networks. *The Journal of Bifurcation and Chaos*, 17(7):2303–2318, 2007.

[5] H.-M. Chen, C. Ming-Hsiu, P.-C. Chang, M.-C. Tien, W. H. Hsu, and J.-L. Wu. Sheepdog - group and tag recommendation for flickr photos by automatic search-based learning. In *ACM Multimedia*, pages 737–740, 2008.

[6] S. Dolev, Y. Elovici, and R. Puzis. Routing betweenness centrality. *The Journal of the ACM*, 57(4), 2010.

[7] S. Golder. Measuring social networks with digital photograph collections. In *Proceedings of the ACM conference on Hypertext and hypermedia (HT)*, pages 43–48, 2008.

[8] R. A. Hanneman and M. Riddle. *Introduction to social network methods*. http://www.faculty.ucr.edu/∼hanneman/nettext/, 2005.

[9] M. J. Huiskes, B. Thomee, and M. S. Lew. New trends and ideas in visual concept detection: The mir flickr retrieval evaluation initiative. In *MIR '10: Proceedings of the 2010 ACM International Conference on Multimedia Information Retrieval*, pages 527–536, New York, NY, USA, 2010. ACM.

[10] B. Marshall. Taking the tags with you: Digital photograph provenance. In *Proceedings of the IEEE International Symposium on Data, Privacy, & E-Commerce*, pages 72–77, 2010.

[11] B. Marshall. Context seeking with social tags. In *Proceedings of the ACM International Conference on Knowledge Management Workshop Exploiting Semantic Annotations for Information Retrieval*, pages 11–12, 2011.

[12] B. Marshall and S. Pandey. Using the social networking graph for image organization. In *Proceedings of the International Conference on Advances in Future Internet*, pages 102–107, 2010.

[13] H. Mller, P. Clough, T. Deselaers, and B. Caputo. *Experimental Evaluation in Visual Information Retrieval*. The Information Retrieval Series - Springer, 2010.

[14] K. Okamoto, W. Chen, and X.-Y. Li. Ranking of closeness centrality for large-scale social networks. In *Proceedings of the ACM International Workshop on Frontiers in Algorithmics*, pages 186–195, 2008.

[15] T. Rattenburg, N. Good, and M. Naaman. Towards automatic extraction of event and place semantics from flickr tags. In *Proceedings of the ACM SIGIR Conference on Research and development in information retrieval*, pages 103–110, 2007.

[16] B. Sigurbhornsson and R. van Zwoi. Flickr tag recommendation based on collective knowledge. In *Proceedings of ACM WWW*, pages 327–226, 2008.