SOCIOPART: Hopping Based Partitions

by

Vivek Mahajan, Lini Thomas, Kamalakar Karlapalem

Report No: IIIT/TR/2009/183

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2009
ABSTRACT
Online social networking has become a phenomenon in the last few years. The growing usage of social networks has put the vendors under a lot of pressure to efficiently process the users queries. In social networks like orkut, most of the queries access the immediate neighbors of the current node and neighbors of its neighbors. Existing graph partitioning approaches focus on reducing the number of inter server links, neglecting the query patterns. Our algorithm focuses on partitioning the graph so as to minimize the number of servers to be accessed in order to process a query. Our results show that the proposed algorithm gives useful partitioning of social networks, thereby reducing inter server accesses.

1. INTRODUCTION
In the last few years, the online social networks(OSNs) have grown in huge proportion with respect to the number of users as well as its utility. The amount of usage of OSNs exceeds e-mail usage[1] which is remarkable. OSNs are the most popular sites on the Internet[9]. A study reported that OSNs account for 10% of all the Internet time worldwide. The growth can be appreciated by the fact that there was 1382% increase in the user base of twitter in one month[Feb-Mar 09] [2]. There can be variations of OSNs in terms of usability e.g On orkut, the users interact with each other by exchanging scraps or photographs, while on SLASHDOT, the users interact via blogs. Facebook, LinkedIn are other examples of OSNs.

The phenomenal growth in the size of OSNs has given rise to scalability issues that can be solved by distributing online social network graph among different servers. However the basis of partitioning is a major issue. That is, the graph partitioning could be based on structural, geo-political or traffic properties, which are discussed in detail in [18] for twitter and orkut. In this paper, we introduce a new partitioning criterion called hop based partitioning which takes into consideration the typical nature of social network queries.

1.1 Hop Based Partitioning
Social network queries typically visit their one hop and two hop neighbors. The small world phenomenon links any two users on a social network with a distance of six hops. Hence, while partitioning a social network, it would be beneficial to maintain the one-hop and two-hop neighbors of a frequented query node on the same server\(^2\) as itself. Further, it is expected that any graph partitioning algorithm attempts to reduce the inter server edges and equally distribute the data across servers.

For example, [14] analyzed large-scale traces of information dissemination in the Flickr social network. Flickr allows users to create two types of links: links to favorite photos (called favorites in Flickr) and links to other users (called contacts in Flickr). Users may favorite-mark a photo and share interesting photos with others. [14] refers to users who include a photo in their favorite photos list as fans of that photo. It was observed that for less popular photos, 91% of all fans are within two hops of the uploaders. Further, for even for top popular photos, 81% of all fans are within two hops of uploaders. [16] mentions that most information is found one-hop away from the query node. According to a study[15], the maximum distance between two persons is six hops. Hence, it is a reasonable assumption that a user interacts primarily within his 2-3 hop radius and at most his six hop radius.

In this paper, we design an algorithm SOCIOPART that caters to the hop based partitioning approach by maximizing the probability of a query node having its one and two hop neighbors on the same server as itself. Given a social network graph G, SOCIOPART partitions the graph while meeting the following constraints:

a. For any node in a partition most of its two-hop neighbors lie within the same partition.
b. The inter partition edges are minimized.
c. All partitions are approximately equally sized.

The paper is organized as follows. The related works is shown in section 2, the evaluation index, We describe our algorithm SOCIOPART in section 3 and the evaluation criterion are formulated in section 4 and experimental results are shown in section 5. Finally, we conclude the paper in section 6.

2. RELATED WORK
The study of graph partitioning algorithms based on different properties of social networks has been proposed in [18]. In the recent past, graph partitioning for dealing with
scalability issues and handling large datasets has been proposed in [7, 8].

With the increased usage of social networks, the scientists have become more interested in analyzing the properties of social networks. Sociologists showed that the average path distance between two Americans is 6 hops in [15] and small world effect analysis was given in [17]. Study of an online social network at Stanford university finds the small world behavior and local clustering property in [13]. The small world property for offline social networks has been discussed in [9,10,11] and it was verified for the online social network in [4, 24]. Analysis of different structural and scale free properties are also extensively analyzed in [4].

The traditional graph partitioning algorithms[19, 22, 21] partition the graph into roughly equal sized partition and focuses on minimizing the edge cuts. Further, the sociological approaches like hierarchical clustering based partitioning and natural division based partitioning[6] approaches are popularity known. Another genre of graph partitioning algorithms that focus on producing quality partitions in less time are multilevel algorithms, which include METIS[11] and its modification for power law graphs[3]. There are other partitioning algorithms like Kernighan-Lin[12] and it’s linear time heuristics[5] can also be used as refinement algorithms.

In this paper, we compare our results with METIS[11]. METIS focuses on minimizing the cuts between the partitions as well as into roughly equal partitions. The METIS algorithm is divided into three phases termed as coarsening, partitioning and uncoarsening/refinement. The main motivation behind the algorithm is to generate an approximate graph of the original graph, partitioning it and then retracting the coarsening phase while doing the refinement between each iteration. SOCIOPART also follows a hierarchical clustering approach where repeated merging of clusters is done until a cluster cannot be further expanded due to a imposed size limit. Finally, the refinement algorithm is applied on the clusters thus obtained. We differ from METIS, in terms of our partitioning criterion, our novel way of clustering and partitioning, and the modification and the method of refinement applied which is applied only once.

3. SOCIOPART

In this section, we introduce our graph partitioning algorithm. A social network partitioning algorithm is expected to reduce inter server accesses requiring neighbors and the neighbors of neighbors of a query nodes to be present in the same server as itself. neighbors are assumed to be queried upon while next queried set is the neighbors of neighbors. To reduce the partitioning overhead, it is important that inter server links be reduced while trying to maximize the number of nodes for which its one and two hop neighbors be on the same server as itself. Further, in order to equally distribute the workload across servers and optimally use the servers used, it is essential that the partitions be approximately same sized. Our algorithm attempts to fulfill efficiently the above requirements while attempting to maximize the number of nodes for which a majority of its one-hop neighbors and with next priority its 2-hop neighbors are included within the same server. Our algorithm is divided into two phases as can be seen in Figure 1, namely the partitioning phase and the refinement phase. The partitioning phase makes an approximate partition of the graph attempting to club all neighbors and neighbors of neighbors of nodes into a single partition. A weight function is used in order to decide which nodes be given priority during the partitioning phase. The refinement phase next reduces the inter partition edges by making slight modifications to the partitioned graph. We next give definitions of commonly used terms followed by a detailed description of each of the phases.

3.1 Definitions

Social Network Graph : The users in the social network form the nodes of the graph while the relationships between the users form the edges of the social network graph.

k-hop neighbor : A node is said to be a k-hop neighbor of a query node if there are K-edges separating the node from the query node. A one hop neighbor can also be a two hop neighbor and so on in case of triangles present in the graph.
Algorithm: SOCIOPART

Input: Graph, No of Partitions, Limit

Phase 1 - Graph Partitioning
1. Removing one degree ()
2. Node weight calculation() - Calculate the weights for each node according to the algorithm discussed in section 3.2
3. Clustering() - Forming clusters according to the sorted order of the weights.
4. Graph formation from clusters() - Consider a cluster as a node for the graph and collapse the edges to form a graph
5. Repeat step 1-4 until the number of nodes in the graphs is equal to the number of partitions or there can be no more cluster formation as any cluster formation will result into number of nodes in a partition to exceed the limit.
6. Form required number of partitions from the remaining nodes by greedily picking the nodes in a cluster.

Phase 2 - Refining
Refine the partitions using kenighan-Lin algorithm or the modified one.

Table 1: Algorithm SOCIOPART

<table>
<thead>
<tr>
<th>Algorithm: SOCIOPART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Graph, No of Partitions, Limit</td>
</tr>
<tr>
<td>Phase 1 - Graph Partitioning</td>
</tr>
<tr>
<td>1. Removing one degree ()</td>
</tr>
<tr>
<td>2. Node weight calculation() - Calculate the weights for each node according to the algorithm discussed in section 3.2</td>
</tr>
<tr>
<td>3. Clustering() - Forming clusters according to the sorted order of the weights.</td>
</tr>
<tr>
<td>4. Graph formation from clusters() - Consider a cluster as a node for the graph and collapse the edges to form a graph</td>
</tr>
<tr>
<td>5. Repeat step 1-4 until the number of nodes in the graphs is equal to the number of partitions or there can be no more cluster formation as any cluster formation will result into number of nodes in a partition to exceed the limit.</td>
</tr>
<tr>
<td>6. Form required number of partitions from the remaining nodes by greedily picking the nodes in a cluster.</td>
</tr>
<tr>
<td>Phase 2 - Refining</td>
</tr>
<tr>
<td>Refine the partitions using kenighan-Lin algorithm or the modified one.</td>
</tr>
</tbody>
</table>

Figure 2: Probability based weight calculation

The algorithm applied that repeatedly applies the weight function to cluster the clusters of the previous level until a given cluster exceeds a predefined size. The parts of the partitioning phase has been diagrammatically shown in 1. The algorithm first clubs all one-degree nodes to its immediate neighbors. It next finds seed nodes which are used as the central points for the clustering phase. A new graph is generated where each of the formed clusters are now represented as nodes with inter partition edges forming edges of the generated graph. The process is then repeated until the required number of partitions are formed.

- Graph Definition: Graph \( G=(V,E) \), where \( V = \{V_1, V_2, \ldots, V_n\} \). \( V_i \) represents a cluster of nodes. In the first level, \( V_i \) consists of only single node and \( E \) is the edge set which consists of \( E_{ij} \) i.e the number of connection between the \( V_i \) and \( V_j \) i.e for any \( V_i \) and \( V_j \), if there are \( K \) edges between the two then \( E_{ij} \) will be equal to \( k \), for the first level \( E_{ij} = 1 \) or \( 0 \) as \( V_i \) and \( V_j \) represents single node.

- Node Weight Calculation: In this paper, we adopt two methods for assigning weights to nodes. Nodes of higher weight values are later picked as seeds during the process of clustering. The two weight functions are: first probability based and the other two-hop based.

  - Probability based weight function: For a query node, let \( S \) be the set of the one hop neighbors of the query node along with the query node itself. The weight function assigns a high weight to the query node if a high number of the one-hop and two-hop neighbors of the nodes in set \( S \) lie within the set \( S \) itself. That is, by forming a cluster of the query node along with its one hop neighbors, we have ensured the presence of one and two hop neighbors of most of the nodes in the cluster. The higher is the weight of the node, the greater is the probability of the node and its neighbor set finding a large number of one and two hop neighbors on the server.

Suppose \( X \) is the set of nodes that form the graph, \( X = \{x_1, x_2, \ldots, x_n\} \) and \( W \) is the set of weights assigned to the set of nodes \( X \). \( W = \{w_1, w_2, \ldots, w_n\} \)

Let the set of mutual neighbors of two nodes \( (n_1, n_2) \) be defined as the set of one hop neighbors of \( n_1 \) that are also one hop neighbors of \( n_2 \). Let the mutual neighbors of \( n_1 \) and \( n_2 \) be denoted as mutual\((n_1, n_2)\). Let neighbors\((x_i)\) denote the set of one hop neighbors of \( x_i \). The node \( x_i \) in the graph at level \( j \), could be a single node or a compound node that was obtained by the merging of clusters at the previous levels. \( |x_i|=1 \) if it is a single node(as in level 1). If \( x_i \) denotes a compound node, \( |x_i| \) denotes all the nodes of the initial graph that have been merged to form the cluster that \( x_i \) denotes.
Removing One Degree Nodes: Nodes of degree one get a weight of zero as per the probability based weight function and are hence ignored. For the two-hop based weight function we assume the single degree nodes will not give rise to many one-hop and two-hop neighbors and are hence ignored. In order to have high degree the single degree node needs to have a high degree neighbor which will have more one and two hop neighbors than the single degree node and would hence take this node into its cluster during the clustering phase. Removal of such single degree nodes thus reduces the computation of the weight function and do not contribute to the clustering phase.

• Clustering Algorithm: The algorithm behaves like a hierarchical clustering algorithm where at each level clusters are formed by clustering the clusters of the previous level. The clustering algorithm varies based on the weight picking function used as described below.

For a given level \(i\), the nodes of the graph at level \(i\) are processed in decreasing order of weight. Initially the input graph forms the input of the clustering algorithm for level one. For each level \(i > 1\), a new graph is generated which is fed as input to the clustering algorithm.

- Probability based clustering: Each selected node assigns all its immediate neighbors to a new cluster. A neighbor that has already been assigned to another cluster in the same iteration of level is ignored. The process is repeated until every node is assigned some cluster number.

- Two-hop based clustering: Each selected node has all its one-hop and two-hop neighbors assigned into one new cluster. Any one-hop or two-hop neighbor already assigned to another cluster in the same iteration is ignored. This process is repeated until every node is assigned some cluster number.

A predefined cluster size is set as: 

\[
\text{Cluster Size} = \frac{\text{Number of Nodes in the initial graph \cdot Number of partitions}}{\text{User defined value \cdot x\% of (Number of Nodes \cdot Number of partitions)}}
\]

where \(x\) is some user defined value of \(x\).

The user defined value \(x\) in the equation above is referred to as the "limit value" through the rest of the paper. The size of any cluster is required to be less than the predefined \(\text{Cluster Size}\). If no node can be selected as a seed node as on selecting it would be giving rise to a cluster that exceeds the \(\text{cluster size}\), then some heuristics applied to cluster the remaining nodes in a graph. A cluster that cannot accommodate further nodes is marked and the new graph formed ignores the node representing this cluster for further clustering.

![Figure 3: Greedy Example](image-url)
picked as seeds causes the formed cluster to exceed the cluster size. It might also so happen that even an individual node cannot be fused with another node due to the cluster size constraint being broken. For example, consider a graph of 100 nodes which requires the bi partition to be done on it. Say, SOCIOPART has so far build three clusters C1, C2 and C3, each of size 33, 33 and 34 respectively as in figure 3. Considering any node as a seed will require C1, C2 and C3 to be merged into a single cluster. Fusing any two nodes without using the clustering algorithm will cause the new cluster to exceed the cluster size of 50.

We give below the sketch of a simple algorithm we use to fix this situation.

**Step 1** - Consider the case when no seed node can be used to form clusters due to an increase in cluster size if the seed is used for clustering. If a n partition of the social network graph is required, we pick the node with the n+1n rank as per the size of nodes. In the order of increasing order of ranks, we fuse the ranked node with another within the first n ranks with which it shares the highest edge weight. This is done for the node within the first n ranks such that on fusing with that node, the cluster size is not exceeded. This process is repeated until no two nodes can be further fused. The next step describes the case when two nodes cannot be fused as per the example above.

**Step 2** - When two nodes cannot be fused, the lowest sized cluster C1 is split back to get the clusters of the previous level of the hierarchy. Each node attained by splitting C1 is then merged with the top n sized cluster nodes by step 1 for the n sized partition.

We see in our experiments that repeating step 1 alone gives us the required partitions with the cluster size maintained.

- **Graph Formation from Clusters**: For the first level before the algorithm starts, each node of the input graph is represented as a cluster in itself. After the first level of clustering at level one, a new graph is generated to act as the input node for level two on which the four processes are repeated. A new graph Gt = (Vt, Et) is formed for the clustering at level i > 1 after the clustering at level i − 1. Each cluster formed at a given level i − 1 is denoted as a new node. Hence the number of clusters at level i − 1 form the number of nodes at level i. The intra cluster edges form the edge set Et of the graph Gt.

### 3.3 Refinement Phase

When the required number of partitions are formed, it gives us an approximate partition of the graph. To refine the partitions, we use a modified form of Kernighan-Lin algorithm[12].

The Kernighan-Lin algorithm starts with initial bipartitions of the graph and then iteratively optimizes it using a greedy approach. The Kernighan-Lin algorithm is divided into two phases, in the first phase, we consider a pair of nodes, with each node from different partitions. Now, a gain value δQ is calculated for all possible pairs, which is the change in the number of edges when the nodes in the pair are swapped. The pair with the maximum gain is marked for swapping so that they are never marked again in this phase. This continues until all the pairs are marked. In the next phase, the algorithm examine the sequence of the swapped pairs and the sequence which results into the maximum gain is selected and the actual swapping takes place. This results into a locally optimized graph and the two phases are repeated till there is no further gain in the edges between the partitions.

We modify one of the phases of Kernighan-Lin algorithm for our criterion. To be precise, we change the definition of δQ for us. In the Kernighan-Lin algorithm, δQ is the change in the edges cuts between the partitions when the nodes are swapped. Let a, b be two nodes that lie in different partitions. Let Ia be the internal cost for a, i.e the neighbors of a that lies in the same partition as of a and Ea be the external cost for the node a, i.e the number of neighbors of a that lie in the separate partition. Similarly Ea and Ia are the external cost and internal cost for the node b. Now let Da = Ea + Ia and Db = E_b + I_b, i.e the difference of external and internal costs. If a and b are interchanged then the gain i.e δQ will be Da + Db − 2C_{a,b}. We make changes in the internal, external and C_{a,b} to fit our purpose of decreasing the probability of the two hops from one node lying in the different partitions. We define internal cost D_{a,b} as the probability of an edge of node Ea, Ea lying in the same partition as of a ,external cost as probability of an edge of node a lying in a separate partition. Hence,

$$I_a = \frac{I_a}{\text{Degree of node } a}$$

and

$$E_a = \frac{E_a}{\text{Degree of node } a}$$

$$2C_{a,b} = \frac{1}{\text{Degree of node } a} + \frac{1}{\text{Degree of node } b}$$

As the original Kernighan-Lin algorithm focuses on reducing the edges between the partitions, our modified one reduces the probability of one hop of the nodes to lie in the separate partition. The rest of the kernighan-Lin algorithm works in the original way. Experiments shows that it not only reduces the probability of one hop to lie outside the partitions but also decreases the edge cuts. The same approach could be applied to the linear heuristics from the felduccia and matthews algorithm[5] which can make it more efficient in terms of time taken to refine the partitions.

### 4. EVALUATION CRITERIA

In this section we will discuss our evaluation criteria. We introduce here two evaluation indexes. In the first evaluation index i.e the probability index, we assume that the inter server costs are high. Consider that a first hop neighbor of a query node lies on another partition, whose immediate neighbor lies on the same partition as the query node. Due to high inter server costs, we would be paying high costs to access the second hop neighbor as well. This is because one needs to access another partition and from there the initial partition again causing two switches in servers. Hence, we assume that the cost of the two hop neighbor is as bad
or worse than its one hop neighbor though present on the same partition as the query node. In the second evaluation index, hop index, we only accumulate the number of first and second hop neighbors that lie outside the initial partition.

- **Probability Index**: For this evaluation index, we assume that there is an equal chance of each user of an OSN to come online. Given some partitions, an evaluation criteria should be able to give a good index of the quality of partitions in terms of fraction of one and two hops lying outside the partition and also should give importance to the random nature of the users of OSNs. If a user is randomly selected, the probability of two hops lying outside its partitions is a good evaluation criteria as it is not biased towards any user and gives a precise value. As we assume that for a node whose two hop lies outside we check probability of error based on whether its one hop lies outside and whether its one hop though lying inside, the two hop lies on a different partition. As we assume that the one hop from the neighbors which are in the separate partition will remain in that partition, so the probability of the one hop and two hops via that neighbor will be equal to the probability of its being in the separate partition. Formally,

Let $X = \{ X_1, X_2, X_3, \ldots, X_n \}$ be the nodes in an OSN.

$X_i = i^{th}$ node

$X_{ij} = j^{th}$ neighbor of the $i^{th}$ node

$N_{Neighbors, \text{different}} = \text{Number of neighbors of } X_i \text{ lying in the separate partition}$

$P_{X_i}^{1\text{-hop}} = \text{Probability of one-hop neighbor of } X_i \text{ to lie outside.}$

$P_{X_i}^{1\text{-hop}} = \frac{N_{Neighbors, \text{different}}}{\text{Number of neighbors of } X_i}$

$P_{X_i}^{2\text{-hop}} = \text{Probability of two hops from } X_i^{th} \text{ node to lie in the separate partition}$

$P_{X_i}^{2\text{-hop}} = \frac{1}{\text{Number of neighbors of } X_i} \sum_{i=0}^{n} P_{X_{ij}}^{1\text{-hop}}, \text{ n is the number of neighbors of } X_i$

where, $P_{X_{ij}}^{1\text{-hop}} = \frac{1}{\text{Number of neighbors of } X_i} * P_{X_j}^{1\text{-hop}}$, if $X_i$ and $X_j$ are in the separate partition

$P_{X_{ij}} = \text{Probability of one hop from } X_i \text{ to } X_{ij} \text{ to lie in the separate partition}$

$P_{X_{ij}} = \frac{1}{\text{Number of neighbors of } X_i}$ if

$X_i$ and $X_j$ are in the separate partition

$\text{Probability Index} = \frac{1}{\text{Total number of nodes}} \sum_{i=0}^{n} P_{X_i}^{2\text{-hop}}, \text{ n is the number of nodes}$

This evaluation index is overall probability of the two hops to lie in the separate partition.

- **Hop Index**:

In this criteria, a percentage of total one hops and two hops lying outside is calculated for each node and an average of this value gives the value for second evaluation index. An error for a node consists of two values, the first is the fraction of one hops from that node lying outside the partition and second one is the fraction of two hops lying outside the partition. The summation of these two values will give the total error rate for one node. The averages of the error rate will give the first evaluation index. Formally, for $i^{th}$ node $X_i = \text{Total number of one hops from the node lying in the separate partition from that node}$

$One\_hop\_error_i = \frac{X_i}{\text{Total number of one hop from } i^{th}\text{node}}$

$Y_i = \text{Total number of two hops from the node lying in the separate partition from that node}$

$Two\_hop\_error_i = \frac{Y_i}{\text{Total number of two hops from } i^{th}\text{node}}$

$Total\_error_i = One\_hop\_error_i + Two\_hop\_error_i$

$Hop\_Index = \frac{\sum_{i=1}^{n} \text{Total\_error}_i}{\text{Total\_number\_nodes}}$

Note that a node can be both a one hop neighbor as well as a two hop neighbor of a query node. In such a case, the node is considered only in the set $X$ of one hop neighbors of the query node and not in the set $Y$ of two hop neighbors in the One\_hop\_error and Two\_hop\_error calculation above. These are the two evaluation criteria based on which we will compare results with METIS algorithm. The first evaluation index is also the motivation for the probability based seed picking and for the modified kernighan Lin refinement algorithm.

### Table 2: Two Partitions - Evaluation Indexes

<table>
<thead>
<tr>
<th>DataSets</th>
<th>Probability based</th>
<th>Hop Index</th>
<th>Two-hop based</th>
<th>Probability Index</th>
<th>Hop Index</th>
<th>METIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOC-Epinions</td>
<td>0.315977</td>
<td>0.304285</td>
<td>0.139007</td>
<td>0.316676</td>
<td>0.162591</td>
<td>0.313931</td>
</tr>
<tr>
<td>SLASHDOT</td>
<td>0.378277</td>
<td>0.496867</td>
<td>0.347178</td>
<td>0.444758</td>
<td>0.438342</td>
<td>0.762978</td>
</tr>
<tr>
<td>Facebook</td>
<td>0.225316</td>
<td>0.273173</td>
<td>0.235946</td>
<td>0.266732</td>
<td>0.21507</td>
<td>0.269943</td>
</tr>
</tbody>
</table>

### Table 3: Three Partitions - Evaluation Indexes

<table>
<thead>
<tr>
<th>DataSets</th>
<th>Probability based</th>
<th>Hop Index</th>
<th>Two-hop based</th>
<th>Probability Index</th>
<th>Hop Index</th>
<th>METIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOC-Epinions</td>
<td>0.283342</td>
<td>0.545528</td>
<td>0.357649</td>
<td>0.592135</td>
<td>0.330739</td>
<td>0.64517</td>
</tr>
<tr>
<td>SLASHDOT</td>
<td>0.519287</td>
<td>0.748442</td>
<td>0.545494</td>
<td>0.825026</td>
<td>0.474338</td>
<td>0.777639</td>
</tr>
<tr>
<td>Facebook</td>
<td>0.406772</td>
<td>0.709073</td>
<td>0.364512</td>
<td>0.546416</td>
<td>0.287844</td>
<td>0.409927</td>
</tr>
</tbody>
</table>
5. EXPERIMENTAL RESULTS

We have picked three popular and large datasets from our experimental results.

1. Epinions Social Network - This is who-trust-whom online social network of a general consumer review site Epinions.com. Members of the site can decide whether to "trust" each other. All the trust relationships interact and form the Web of Trust which is then combined with review ratings to determine which reviews are shown to the user. The dataset contains 75879 nodes and 508837 edges and was first used in [20]. One can download the dataset from the link\(^3\).

2. Slashdot Social Network - Slashdot is a technology-related news website known for its specific user community. The website features user-submitted and editor-evaluated current primarily technology oriented news. In 2002 Slashdot introduced the Slashdot Zoo feature which allows users to tag each other as neighbors or foes. The network contains friend/foe links between the users of Slashdot. The dataset consists of 77360 users and 905468 edges. The network was obtained in November 2008 and used in [10]. It can be downloaded from the link\(^4\).

3. Facebook Social network - Facebook is a true social networking site on which user interact by sending messages, sharing pictures etc. The dataset was collected during December 29th, 2008 and January 3rd, 2009 and during January 20th 2009 and January 22th 2009. The original dataset is time stamped, mentioning when were the links made. For our purposes we used it as an undirected graph giving no consideration to time stamp attribute. The dataset was crawled for [23] for analyzing the evolution of interaction between the users.

We compare the probability based partitioning approach and the two hop based partitioning approach with METIS. We have three criterion on which we are comparing the results. Two of them are the evaluation indexes i.e probability index and hop index. Probability index is the probability based evaluation index while hop index is the two hop based evaluation index. The third criterion is the edge cuts between the partitions. The lesser the value of each of these indexes, the better the algorithm performs according to each criterion. We have generated results for two partitions and three partitions for the three real life datasets. Variations of evaluation indexes with respect to two partitions and three partitions are shown in Table 2 and 3. There are total number of four columns in each table, the first column contains the names of datasets, the next three columns are for three algorithms i.e probability based, two hop based and METIS. Each of these columns are further divided into two columns, with the first representing the probability index and second hop index. Comparison of the edge cuts for two and three partitions are shown in Table 4 and 5. Due to the limitations on the paper size, we present our results for the limit value 0.6 while comparing the algorithms. The figures 4-15 however give us a idea of the variation of evaluation index values with varying limit size.

### Table 4: Two Partitions - Edge cuts

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Probability based</th>
<th>Two-hop based</th>
<th>METIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soc-Epinions</td>
<td>50791</td>
<td>47400</td>
<td>58104</td>
</tr>
<tr>
<td>Slashdot</td>
<td>123390</td>
<td>105671</td>
<td>82685</td>
</tr>
<tr>
<td>Facebook</td>
<td>63299</td>
<td>68885</td>
<td>75042</td>
</tr>
</tbody>
</table>

### Table 5: Three Partitions - Edge cuts

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Probability based</th>
<th>Two-hop based</th>
<th>METIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soc-Epinions</td>
<td>92999</td>
<td>122876</td>
<td>81392</td>
</tr>
<tr>
<td>Slashdot</td>
<td>194005</td>
<td>207971</td>
<td>153838</td>
</tr>
<tr>
<td>Facebook</td>
<td>259822</td>
<td>172856</td>
<td>106015</td>
</tr>
</tbody>
</table>

\(^3\)http://snap.stanford.edu/data/soc-Epinions1.html

\(^4\)http://snap.stanford.edu/data/soc-Slashdot0811.html

**Soc-Epinions Social Network**
- **Two partitions** - For probability index i.e the probability based evaluation index, the probability based algorithm is doing remarkably better than METIS and comparable to Two-hop algorithm while again for the two-hop based index i.e hop index, probability based algorithm is doing better. For the third criterion i.e the edge cuts, the probability based and two hop based algorithm is performing remarkably better than METIS.
- **Three partitions** - For probability index, again the probability based algorithm is giving the best results and also for the hop index. The edge cuts are comparable with the edge cuts formed by the METIS algorithm.

**SLASHDOT**
- **Two partitions** - For probability index the two-hop based algorithm is giving better results which is comparable to the value of probability index in probability based, while for hop index, again two hop based algorithm is doing remarkably better than the METIS. In terms of edge cuts, METIS is giving lesser edge cuts comparatively.
- **Three partitions** - METIS is giving better value for probability index while probability based algorithm is giving better value for hop index. In terms of edge cuts METIS is giving better results.

**Facebook**
- **Two partitions** - Both evaluation indexes for all the algorithms are comparable, while there is a remarkable difference between the edge cuts of METIS and the probability based algorithm.
- **Three partitions** - METIS does comparatively better in all the three criterion.
We further do an analysis of the limit value that is required as a user input for the SOCIOPART algorithm. We study the change in the evaluation index value and the edge cut size with variations in the limit value in Figures 4 to Figure 15. It can be seen that there is no definite pattern of change in both evaluation index values for both algorithms for both bipartitions as well as three partition experiments. However, the edge cut shows a slow decline with increasing limit values.

It has been seen that the two hop and probability based method does better in a majority of cases. We present the results of the two hop partitioning algorithm in order to be able to compare the naive strategy of accommodating two hops on the same server as against the probability based approach. The SOCIOPART algorithm takes 2-3 mins to run on all the datasets. SOCIOPART essentially contributes in providing a probability based partitioning algorithm based on the query access patterns shown by users in social networks.

6. CONCLUSION

The number of users on the online social networks have increased phenomenally. Existing algorithms do not give priority to the pattern of access by the query node. We propose a partition criterion based on the small world phenomenon where most social network users tend to access their immediate and two hop neighbors. Our query based approach to partition the online social networks reduces the cost of access by maximizing the number of nodes for which most of its one and two hop neighbors lie on the same server as itself. In our future work, we plan to extend the partition criterion to be extended to k-hops being present on the same server for frequent query nodes.

7. REFERENCES

Figure 8: Probability based bipartitions

Figure 9: Two-hop based bipartitions

Figure 10: Probability based three partitions

Figure 11: Two-hop based three partitions


Figure 12: Two hop based three partitions

Figure 13: Probability based three partitions

Figure 14: Probability based three partitions

Figure 15: Two hop based three partitions

