CRITICAL CLIQUES AND THEIR APPLICATION TO INFLUENCE MAXIMIZATION IN ONLINE SOCIAL NETWORKS

A Thesis

by

NIKHIL PANDEY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2012

Major Subject: Computer Science

CRITICAL CLIQUES AND THEIR APPLICATION TO INFLUENCE MAXIMIZATION IN ONLINE SOCIAL NETWORKS

A Thesis

by

NIKHIL PANDEY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Jianer Chen
Committee Members,	Donald K. Friesen
	Jennifer L. Welch
	Catherine Yan
Head of Department,	Duncan M. Walker

May 2012

Major Subject: Computer Science

ABSTRACT

Critical Cliques and Their Application to Influence Maximization in Online Social Networks. (May 2012)

Nikhil Pandey, B.Tech., Motilal Nehru National Institute of Technology

Chair of Advisory Committee: Dr. Jianer Chen

Graph decompositions have useful applications in optimization problems that are categorized as NP-Hard. Modular Decomposition of a graph is a technique to decompose the graph into non-overlapping modules. A module M of an undirected graph $G = (V, E)$ is commonly defined as a set of vertices such that any vertex outside of M is either adjacent or non-adjacent to all vertices in M. By the theory of modular decomposition, the modules can be categorized as parallel, series or prime modules. Series modules which are maximal and are also cliques are termed as *simple series modules* or *critical cliques*.

There are modular decomposition algorithms that can be used to decompose the graph into modules and obtain *critical cliques*. In this current research, we present a new algorithm to decompose the graph into *critical cliques* without applying the process of modular decomposition. Given a simple, undirected graph $G = (V, E)$, the runtime complexity of our proposed algorithm is $O(|V| + |E|)$ *under certain input constraints*. Thus, one of our main contributions is to propose a novel algorithm for decomposing a simple, undirected graph directly into *critical cliques*.

We apply the idea of *critical cliques* to propose a new way for solving the influence maximization problem in online social networks. Influence maximization in online social networks is the problem of identifying a small, initial set of influential individuals which can influence the maximum number of individuals in the network. In this research, we propose a new model of online social networks based on the notion

of *critical cliques*. We utilize the properties of *critical cliques* to assign parameters for our proposed model and select an initial set of activation nodes. We then simulate the influence propagation process in the online social network using our proposed model and experimentally compare our approach to the greedy algorithm proposed by Kempe, Kleinberg and Tardos. Our main contribution in the influence maximization research is to propose a new model of online social network taking into account the structural properties of the social network graph and a new, faster algorithm for determining the initial set of influential individuals in the online social network.

To my parents who have always provided me words of encouragement and support

ACKNOWLEDGMENTS

Special thanks to Dr. Jianer Chen who has been a great advisor and a source of constant knowledge, encouragement and support.

Also, thanks to Jie Meng, Tanushree Mitra and Abhishek Soni for their input in this research.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Critical Cliques

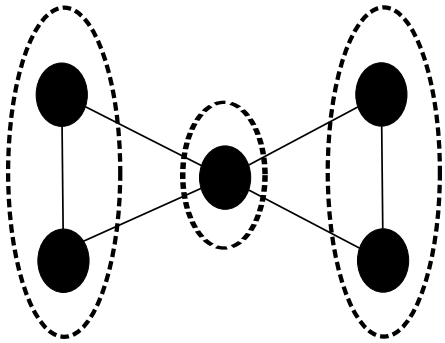

Fig. 1. Critical cliques.

A *critical clique* C of an undirected graph $G = (V, E)$ is a clique such that all the vertices $v \in C$ are adjacent to the exact same set of vertices in G (*same closed neighborhood property*) and C is maximal under the *same closed neighborhood property*. In Figure 1, the dotted ovals represent the *critical cliques* of the shown graph.

The inherent structure of *critical cliques* has the *clique* property as well as that of being a *module* property. These two combined properties make *critical cliques* interesting graph structures with applications to graph related problems. Consider the following application (related to the kernelization for the problem of *Cluster Editing*) discussed in [1]. The problem of *cluster editing* asks if a given graph $G = (V, E)$ along with a parameter $k \geq 0$, can be transformed into a union of vertex-disjoint cliques (cluster graph) by atmost k edge deletions and insertions. Since atmost k

---

The journal model is *IEEE Transactions on Automatic Control*.

edge modifications are allowed, atmost $2k$ vertices in G will be touched (or modified). Therefore, to obtain a kernel for *cluster editing*, the main question revolves around obtaining an upper bound on the vertices that are untouched by edge modifications. This can be done by observing that in the final solution graph, the set of vertices untouched by edge modifications and which belong to the same clique would form a *critical clique* in G. In addition, we can also observe that *critical cliques* of large sizes along with their neighbors should be preserved in the *cluster graph*. This makes it easier to study this problem in the context of *critical cliques* (by dividing the original input graph G into *critical cliques*) and then obtaining reduction rules for the kernel of the *cluster editing* problem.

Based on the notion of *critical cliques*, we have 4k [2] and 2k [3] kernel for the fixed parameterized tractable version of the *cluster editing* problem. As mentioned in [3], the *cluster editing* problem has application in areas like computational biology and machine learning. In this research, we also propose a novel application of the *critical cliques* to the problem of influence maximization in online social networks. We believe that *critical cliques* have more widespread applications than those that have been discovered so far and an algorithm to identify *critical cliques* directly (as proposed in this research) will facilitate their use in further different areas of research.

B. Influence Maximization Problem

We present an application of the *critical cliques* algorithm in the context of influence maximization problem in online social networks. An online social network can be represented as a graph where the vertices in the graph represent the individuals which are a part of the social network and an edge between two vertices signifies a form of relationship between the two corresponding individuals. Such edges (or links)

between nodes can be thought of as carriers of information from one individual to the other.

In recent years, there has been extensive research related to social network analysis. One category of such research has focussed on the question of how social networks can be used to efficiently disperse information across the entire network. Influence Maximization in online social networks is the problem of identifying a small set of influential individuals which when influenced can help maximize the spread of influence (or ideas/information/strategies) across the entire online social network. Identifying such a set of individuals has been shown to be NP-Hard [4].

Influence Maximization problem is greatly significant in the context of word of mouth propagation of ideas and influences across the social network. Consider an example discussed in [5] and [6]. A small company wants to market their online application to users in an online social network like Facebook or MySpace. The company has a limited budget and it wants to target a few users in the online social network and allow them the access and use of their online application for free. What the company hopes in this scenario is that the initial set of selected users will tell their friends about this online application and might influence them to sign up for it. In turn, the influenced friends of the initial set of targeted users will influence their other friends and so on. This could possibly lead to a trigger of a large number of signups for the online application which would ultimately prove to be beneficial for the company. The problem that the company would face is to determine the initial set of target users who could lead to a large number of signups for their application and thus, maximize the influence spread across the social network. This example shows that the problem of influence maximization is important in the area of *viral marketing* based on word of mouth propagation of ideas in online social networks.

CHAPTER II

RELATED WORK

In the current literature, modular decomposition is also known as substitution decomposition, disjunctive decomposition or X-join decomposition. The origin of the theory of modular decomposition is usually attributed to the work of Gallai in [7]. In his work, Gallai introduced the idea of a modular decomposition tree which is a tree representing the modules of an undirected graph. Since the work of Gallai, modular decomposition has found a wide variety of combinatorial applications in the context of graph theory. The modular decomposition tree can be utilized using the *Divide and Conquer* technique for solving a variety of hard problems if the graph can be decomposed non-trivially [8]. Families of graphs such as cographs [9] or permutation graphs [10] can be identified by studying their modular decomposition tree. In the area of fixed parameterized tractability, modular decomposition has been used for kernelization of several problems, such as flip consensus tree [11] or bicluster editing [12].

A number of algorithmic approaches for modular decomposition of undirected graphs have been proposed in the literature. The first algorithm for modular decomposition was proposed in 1972 by Cowan et al. in [13]. This algorithm had a polynomial time complexity of $\mathrm{O}(n^4)$. There were also several proposed algorithms with a runtime complexity of $O(n^3)$ or $O(nm)$ ([14], [15], [16], [17]) and $O(n^2)$ ([18], [19]). Linear time algorithms for modular decomposition first appeared in the 1990s by R.M. McConnell and J. Spinrad. in [20] and also by A. Cournier and M. Habib in [21]. However, these linear time algorithms were complex with not much practical significance. Later on, simplified algorithms with linear runtime([22], [23]) and almost linear runtime([24], [25], [26]) were proposed. For a survey on the algorithms related

to modular decomposition, refer to [27].

In this current research, we are interested in extracting *critical cliques* from an undirected graph without applying the process of modular decomposition. Even though the linear time modular decomposition algorithms can be used to extract *critical cliques*, these algorithms are complex to implement and have a large constant factor [23] which makes the overall process of modular decomposition to extract *critical cliques* expensive. To the best of our knowledge, we are not aware of algorithms that directly extract the *critical cliques* from undirected graphs in the literature as proposed in this current research.

We aim to show an application of the *critical cliques algorithm* to the problem of influence maximization in online social networks. There has been extensive research on the problem of influence maximization in the domain of social computing and social network analysis. This problem was first studied as an optimization problem and shown to be NP-Hard by Kempe et al. in [4]. Kempe et al. also introduced a greedy approximation algorithm with an approximation factor of (1 - 1/e) in [4]. However, the greedy algorithm proposed by Kempe et al. had a significantly large running time even for smaller social network graphs with few thousands of vertices. Since then, there have been various approaches proposed that improve the running time of the greedy algorithm. The problem of influence maximization exhibits the submodular property and Leskovec et al. used this to propose a "Cost-Effective Lazy Forward" (CELF) optimization in [28] to considerably improve the running time of the greedy algorithm in [4]. Other notable approaches are [5], [6] and [29] which propose different heuristics to speed up the running time of the greedy algorithm while not significantly changing the influence spread across the social network.

The greedy algorithm based approaches for the problem of influence maximization in online social networks suffer from a serious drawback: the runtime efficiency of

the algorithm. In addition to this, these approaches do not consider how individuals in the social network are grouped by their common likes or dislikes which can cause the influence of one individual over another to vary. Davis in [30] stated the following:

> [30] Because cliques are defined in terms of high rates of positive ties, adoption of a new attitude or activity by a member will be followed by acceptance within the clique. Because, however, members of other subgroups and isolates have less positive or even negative bonds with members of the innovating group, they will tend to resist or be late adopters.

In this research, we aim to study the problem of influence maximization by decomposing the online social network graph into clusters of individuals such that within each cluster, all the individuals are bound by a form of relationship (e.g. friendship) to each other and have a high degree of similarity with each other. Intuitively, the idea of a graph decomposition into *critical cliques* matches well with the identification of clusters of individuals sharing a high degree of similarity in the social network graph. As far as we are aware, this is the first such application of *critical cliques* to the problem of influence maximization in online social networks.

CHAPTER III

ALGORITHM FOR CRITICAL CLIQUES

A. Definitions

In this research, we consider only simple, undirected graphs. Let $G = (V, E)$ denote a graph where V is the set of vertices and E is the set of edges in the graph G.

- **Open neighborhood of a vertex**

  Open neighborhood of a vertex $v \in V$ is defined to be the set of all vertices that are immediately adjacent to v i.e. all such vertices $w \in V$ such that $(v, w) \in E$. We denote the open neighborhood of the vertex v by $N(v)$.

- **Closed neighborhood of a vertex**

  Closed neighborhood of a vertex $v \in V$ is denoted by $N[v]$. The closed neighborhood of the vertex v is the union of the open neighborhood of v with v itself. Thus, closed neighborhood of v is defined as:
  $N[v] = N(v) \cup \{v\}$

- **Module**

  A module M is defined as a subset of vertices ($M \subseteq V$) such that any vertex $v \in V \backslash M$ satisfies exactly one of the following properties:

  - $(u, v) \in E \ \forall u \in M$

    A vertex v satisfying this property implies that v is adjacent to all the vertices in M.

  - $(u, v) \notin E \ \forall u \in M$

    A vertex v satisfying this property implies that v is adjacent to none of

the vertices in M.

From the above two properties, it is easy to infer that from the perspective of any vertex $v \in V \backslash M$, the vertices in the module M are *indistinguishable* from each other.

- **Strong Module**

  A module M which does not overlap with any other module is termed as a *strong module*.

  Two modules M_1 and M_2 are said to overlap if they satisfy all of the following conditions:

  - $M_1 \cap M_2 \neq \emptyset$

    This property implies that the two modules share common vertices.

  - $M_1 \not\subset M_2$ and $M_2 \not\subset M_1$

    This property implies that no module is contained within the other.

  It is easy to see that in a graph G with n vertices, there cannot be more than $O(n)$ strong modules.

- **Modular Decomposition**

  Modular decomposition of an undirected graph G is a process where the graph is recursively decomposed into a tree-like structure with the nodes of the tree representing strong modules. We denote the modular decomposition tree by $MD(G)$. The root node of the modular decomposition tree $MD(G)$ is the set of vertices V and each vertex $v \in V$ is a leaf node. Note that the nodes in the modular decomposition tree $MD(G)$ represent the complete set of the strong modules of G.

The edge formations in the modular decomposition tree $MD(G)$ can be explained by the recursive construction of $MD(G)$ in the following way: if M is an internal node in $MD(G)$ representing a strong module, then the subtree rooted at M is the modular decomposition tree of the subgraph induced by M i.e. $G[M]$.

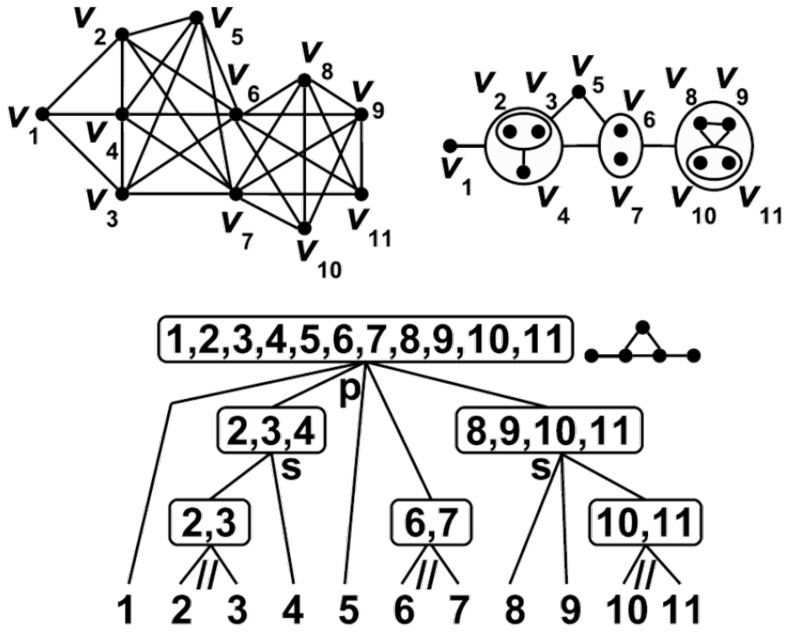

Fig. 2. Modular decomposition of an undirected graph.

(Source: http://en.wikipedia.org/wiki/File:ModularDecomposition.png)

Figure 2 shows the modular decomposition tree for an undirected graph with 11 vertices. Note that the nodes of the tree satisfy the property of being a strong module.

- **Parallel, Series and Prime modules**

The modular decomposition tree of an undirected graph can only contain three different types of modules as stated in the following theorem:

Theorem: [31] Let $MD(G)$ be the modular decomposition tree of

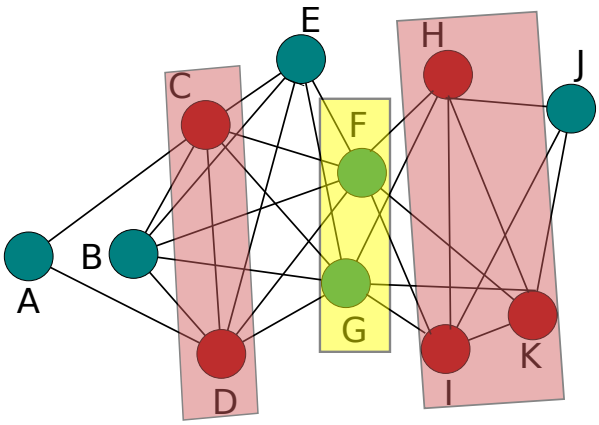

Fig. 3. Types of modules.

graph $G = (V, E)$, and M be an internal node with children $S_1, S_2, ..., S_r$. Then exactly one of the following three cases holds for M, based on the relation of S_i:

- Parallel : There is no edge between S_i and S_j for any $i \neq j$. In Figure 3, {F, G} is an example of a parallel module.
- Series : There is an edge (u, v) between any pair of vertices $u \in S_i$ and $v \in S_j$ for any $i \neq j$. In Figure 3, {C, D} is an example of a series module.
- Prime : None of above two applies to the children.

- **Simple Series Modules or Critical Cliques**

  A *Simple Series Module* or a *Critical Clique* C is a series module with the following properties:

  - For any two vertices $u \in C$ and $v \in C$, $(u, v) \in E$. This is the clique property i.e. any two vertices in C are adjacent to each other.
  - For any two vertices $u \in C$ and $v \in C$, $N[u] = N[v]$, where $N[x]$ denotes the closed neighborhood of a vertex x. This property is because of C being

a series module.

- C should be maximal under the above two properties.

In Figure 3, {C, D} and {H, I, K} are examples of *critical cliques*.

B. Main Algorithm

In this section, we present the algorithm for obtaining *critical cliques* in an undirected graph $G = (V, E)$. We assume that each $v \in V$ is represented by a unique id or number that is assigned from the interval $[0, |V| - 1]$. We use $ADJ(v)$ to denote all the vertices which are adjacent to v (adjacency list of vertex v) and $ADJ_{new}(v)$ to denote the modified adjacency list of vertex v. We allow concatenation to the adjacency list using the plus(+) operator.

The algorithm comprises of three phases, each of which is described as follows:

1. Preprocessing Phase

Algorithm B.1 Preprocess(G)

1: **for all** $v \in V$ **do**
2: $ADJ_{new}(v) \leftarrow [v] + ADJ(v)$
3: $ADJ_{new}(v) \leftarrow sort(ADJ_{new}(v))$
4: $l \leftarrow |N[v]|$
5: $ADJ_{new}(v) \leftarrow [v] + [l] + ADJ_{new}(v)$
6: **end for**
7: max $\leftarrow$ maximum($|ADJ_{new}(v)|$) $\forall v \in V$
8: return $ADJ_{new}(v)$ $\forall v \in V$, max

In this phase, the adjacency list for each vertex $v \in V$ is modified in the following

manner: $ADJ_{new}(v) = [v, |N[v]|, v_1, v_2, v_3....v_k]$, where $|N[v]|$ is the size of the set of the closed neighborhood of v and all of $v_1, v_2, v_3...v_k \in N[v]$.

The order of $v_1, v_2, v_3...v_k$ is in increasing order by their unique ids such that: $v_1 < v_2 < v_3... < v_k$. This is ensured by the *sort*() method used in the algorithm which sorts the list of vertices passed as an input parameter in a non-decreasing order. We can avoid the *sort* step in the *preprocessing phase* in the actual implementation of the algorithm if the adjacency list of all vertices $v \in V$ for the input graph G is already provided in the sorted order.

The *preprocessing phase* returns the modified adjacency list $ADJ_{new}(v)\ \forall\ v \in V$ and another parameter max which is the maximum length of a modified adjacency list amongst all such lists. These two returned parameters are used in the next phase of the algorithm.

2. Sorting Phase

Algorithm B.2 Sorting$(G, ADJ_{new}(v), max)$

1: $L \leftarrow [\ ]$
2: **for all** $v \in V$ **do**
3: Add $ADJ_{new}(v)$ to L
4: **end for**
5: $i \leftarrow 1$
6: **while** $i < max$ **do**
7: $RadixSort(L, i)$
8: $i \leftarrow i + 1$
9: **end while**
10: return L

In this phase, radix sort is performed on the list of modified adjacency lists L. Suppose that the list L at the end of step 4 is constructed as follows:
$L = [ADJ_{new}(v_1), ADJ_{new}(v_2), ... ADJ_{new}(v_n)]$, where $n = |V|$
Except for the first entry in each $ADJ_{new}(v) \in L$, we apply the *RadixSort* procedure using all other elements in $ADJ_{new}(v)$ in step 7, assuming an increasing priority of elements in $ADJ_{new}(v)$ from right to left. Thus, the rightmost element in each of the $ADJ_{new}(v)$ will be the least significant and *RadixSort* procedure will arrange the lists in L using this rightmost element in the first iteration. Note that the second input parameter i to the *RadixSort* procedure signifies that radix sort will be executed using $i'th$ element from the last from each list $l \in L$.

Subsequent iterations of *RadixSort* will consider elements by moving from right to left, one element at a time, upto the second element from the start of the list. It might so happen that a list $l = ADJ_{new}(v) \in L$ gets exhausted during one of the iterations of the *RadixSort* procedure. In such case, l will not be considered for sorting while the other lists are shuffled during that iteration.

At the end of this sorting phase, we would have sorted all the lists in L using *RadixSort* by considering their elements from right to left (except the leftmost element). This list L is returned and used in the next phase of our algorithm.

3. Merging Phase

Since the last element on which radix sort is performed for each list $l = ADJ_{new}(v) \in L$ is the size of the closed neighborhood of v, we note that after the last pass of the radix sort, all the vertices which have equal size of closed neighborhood will be consecutively aligned. In addition, all such vertices v with equal size of closed neighborhood will have the list of vertices in their closed neighborhood in a sorted order in $ADJ_{new}(v)$ (due to the earlier passes of the *RadixSort* procedure).

We now iterate through the list L to obtain all the *critical cliques* in graph G. Let's say u is the first element of $L(1)$ and v is the first element of $L(2)$. It is easy to figure out the *critical cliques* based on the property that two vertices $u \in V$ and $v \in V$ belong to the same critical clique if $N[u] = N[v]$. If $N[u] \neq N[v]$, then u cannot be a part of any other *critical clique* including vertex z, because then the list with first element as vertex z would have appeared above $L(2)$. Thus, in case $N[u] \neq N[v]$, we identify u as a single vertex *critical clique* and compare $N[v]$ to $N[w]$, where w is the first element of $L(3)$. This process is performed until all the lists $l \in L$ are exhausted.

Algorithm B.3 Merge(G, L)

$CC \leftarrow \emptyset$
$T \leftarrow \emptyset$
$l_u \leftarrow L(1)$
$u \leftarrow$ first element of l_u
$T \leftarrow T \cup \{u\}$
for $i = 2 \rightarrow |V|$ **do**
    $l_v \leftarrow L(i)$
    $v \leftarrow$ first element of l_v
    if $l_u = l_v$ from second element to the last element **then**
        $T \leftarrow T \cup \{v\}$
    else
        $CC \leftarrow CC \cup \{T\}$
        $T \leftarrow \{v\}$
    end if
    $l_u \leftarrow l_v$
end for
$CC \leftarrow CC \cup \{T\}$

In the pseudocode for the merging phase, eventually all critical cliques are identified and added to the set CC.

C. Proof of Correctness

We now prove the correctness of our algorithm for finding *critical cliques*. We show that any two vertices $u, v \in V$ belonging to a *critical clique* C will always be grouped together in the same *critical clique*. The proof is by contradiction. Suppose that u and v have been grouped under two different *critical cliques*. As u and v should actually belong to the same *critical clique*, they have the exact same set of closed neighborhood in their modified adjacency lists. Suppose that at the end of the *sorting phase*, the modified adjacency list of u appears before the modified adjacency list of v in the sorted order. There can be two possible cases: either the modified adjacency lists of u and v lie consecutively or they are separated by modified adjacency lists of vertices $w \in V$ which have the same closed neighborhood as u and v. If the closed neighborhood of the vertex w is not the same as that of u and v, then the modified adjacency list of w cannot be an intermediate list as it would violate the sorted order of all the modified adjacency lists. Now in the *merging phase*, the whole group of modified adjacency lists - of vertex u upto vertex v will be merged together in the same *critical clique* by the condition that they have the exact same set of closed neighborhood. This contradicts our initial assumption that u and v were placed in different *critical cliques*.

We also note that the closed neighborhood property guarantees the *clique* property of the *critical clique*. In addition, since any two vertices in the same *critical clique* would always be grouped together, this would guarantee the *maximal* property of the *critical cliques* formed.

D. Complexity

- **Preprocessing phase**

  The main computation in this phase is the sorting of the adjacency list of the vertices. As mentioned earlier, this sorting can be avoided if the input graph G is provided with the adjacency list of vertices already in a sorted order. If this is the case, the computations required to perform in this step would be to create modified adjacency lists for all vertices $v \in V$ by adding $(v, |N[v]|)$ before each adjacency list and adding v in the modified adjacency list maintaining the sorted order. This can be done by creating another empty list, leaving its first two entries empty and copying elements from the original adjacency list in this new list. While copying the elements, we can place v in sorted order in this new list and also maintain a count of the number of elements read and copied from the original adjacency list. At the end of the scan of the original adjacency list of v, we can place $(v, |N(v)| + 1)$ at the beginning of the new adjacency list to create a modified adjacency list for v. The overall runtime complexity of this phase will be $O(|E| + |V|)$ and the space complexity is also $O(|E| + |V|)$ due to the creation of modified adjacency lists.

  In case the adjacency lists of the vertices are not in sorted order, we will need to sort them in the *preprocessing phase*. A good alternative for sorting would be to use a count sort algorithm knowing that the range of unique ids assigned to the vertices are in the interval $[0, |V| - 1]$. A count sort would also work in case we know an upper bound on the vertices of the graph. To sort the vertices using count sort would take $O(|E| + |V|^2)$ running time and $O(|V|)$ space.

  It is a practically feasible assumption to expect the input graph G to be given such that the vertices are assigned unique ids between $[0, |V| - 1]$ and the

adjacency lists for the vertices are provided in a non-decreasing order of these ids. Unfortunately, if this is not the case, we would need to preprocess the graph G before identifying the *critical cliques*. This would incur an extra cost in the *preprocessing phase* due to the sort procedure. For further purposes of calculating the run time complexity, we assume that the input graph G is given in the manner where the vertices have been assigned unique ids and the adjacency lists are in non-decreasing order by their ids.

- **Sorting phase**

  In this phase, the main computation is that the modified adjacency lists for all vertices $v \in V$ are sorted using $RadixSort$ procedure. The implementation of $RadixSort$ procedure for this step can be done by using a hashtable with $|V|+2$ buckets (from 0 to $|V|+1$). In each step of the $RadixSort(L, i)$ procedure, the appropriate pointer to the modified adjacency list $l \in L$ can be placed in the bucket $H(i)$ if the $i'th$ element from the last in list $l = i$. Once all the modified adjacency lists are placed in the buckets, we get a new ordering of all the lists $l \in L$. This new ordering can then be used in the next iteration of the $RadixSort(L, i)$ procedure. In subsequent iterations, we can get the new ordering of the lists by creating another hashtable H', reading all the lists from hashtable H from buckets 0 to $|V|+1$ in that order, placing them in H' based on the current input parameter i, deleting hashtable H and assigning $H = H'$. As mentioned earlier, if any list $l \in L$ gets exhausted in any iteration of the $RadixSort$ procedure, we do not need to touch that list l again and we can continue with the shuffling of other lists. For such an earlier exhausted list l, we can remove it out from the $RadixSort$ procedure while remembering its position in the current bucket in H.

The runtime for placing the modified adjacency lists $\forall v \in V$ in L is $O(|V|)$ (steps 2 to 4). These steps require moving the appropriate pointers for the modified adjacency lists to L. As the modified adjacency list for all vertices $v \in V$ need to be read before the lists are finally sorted, the run time complexity of the *RadixSort* procedure is atleast $O(|E| + |V|)$. We also have an additional complexity of scanning the hashtable H from buckets 0 to $|V| + 1$ in each iteration of the *RadixSort* procedure. Since the iterations run a total of max times, where max is the maximum length among all the modified adjacency list, this additional complexity is: $O(max.|V|)$. We note that max is bounded by the maximum degree d of the graph G.

Thus, the overall runtime complexity for this phase is $O(d.|V|+|E|)$. This phase also requires a space complexity of $O(|E| + |V|)$ due to the use of hashtable.

- **Merging phase**

  In this phase, we compare successive sorted modified adjacency lists $l \in L$ and identify vertices which belong to the same critical cliques. To compare two modified adjacency lists of respective vertices $u \in V$ and $v \in V$, we will have to read both the modified adjacency lists to check for the *critical clique* condition: $N[u] = N[v]$. To facilitate this check, it might be easier to generate a signature for the part of the modified adjacency lists that are being compared. This signature can be a simple string signature formed by concatenating the relevant elements along with a delimiter in the actual implementation.

  The runtime complexity of this phase is $O(|E|+|V|)$ as each modified adjacency list needs to be read atleast once. The space complexity for this phase is $O(|V|)$ which is used for constructing the set CC which is the set of all the *critical cliques*.

- **Overall Runtime Complexity**

  Given that the vertices $v \in V$ of the input graph G have been assigned unique ids in the interval $[0, |V| - 1]$ and the adjacency lists of the vertices of G are provided in sorted non-decreasing order by these ids in the *preprocessing phase*, the runtime complexity of the algorithm is: $O(d.|V| + |E|)$ where d is the maximum degree in G. In case of an input graph G where d is upper bounded by a constant value, the runtime complexity of the algorithm will be linear i.e. $O(|E| + |V|)$.

  Note that there are two main factors contributing to the overall runtime complexity of our algorithm: the factor $d.|V|$ (sorting phase) and scanning the modified adjacency lists in each phase. The constant factor due to scanning the modified adjacency lists of all vertices is small as we scan the modified adjacency lists a total of three times, once during each phase of the algorithm.

- **Space Complexity**

  The additional space required by this algorithm is: $O(|V| + |E|)$.

E. Summary

We proposed a new algorithm for decomposition of a simple, undirected graph $G = (V, E)$ into *simple series modules* or *critical cliques*. The key idea of our algorithm utilized the property of *critical cliques* that they are maximal cliques under the condition that any two vertices in a *critical clique* share the same closed neighborhood. Using this property, we first assigned each vertex a unique integer id and sorted the set of closed neighborhood of each vertex $v \in V$ on those ids. Then, we applied a radix sort procedure on the closed neighborhood of each vertex v, with the last iteration of the radix sort being performed on the size of the closed neighborhood

of v i.e. $|N[v]|$. These steps ensured that the vertices in the same critical cliques will be aligned successively and using a linear scan of $N[v]$ ($\forall v \in V$) in the sorted order, we determined the *critical cliques* of the input graph G.

The runtime complexity of our algorithm is determined to be $O(d.|V| + |E|)$. The space complexity of the algorithm is calculated to be $O(|V|+|E|)$. Note that the runtime complexity is linear under the constraint that the adjacency lists of vertices in the given input graph are sorted in non-decreasing order based on the unique integer ids assigned to the vertices and the maximum degree of the graph is bounded by a constant.

CHAPTER IV

INFLUENCE MAXIMIZATION IN ONLINE SOCIAL NETWORKS

A. Influence Maximization Problem

An online social network can be represented as an undirected graph $G = (V, E)$ where the vertices $v \in V$ represent the individuals present in the social network while an edge $(u, v) \in E$ represents a relationship between the individuals $u \in V$ and $v \in V$. Influence maximization in online social networks is the problem of identifying a small set of individuals in G which when influenced can further influence the maximum number of individuals in the network and thus, help in maximizing the influence spread. This problem has been shown to be NP-Hard with a reduction from the known *SetCover* problem [4].

In [4], Kempe et al. presented a greedy hill climbing algorithm for the influence maximization problem. A general outline of their algorithm discussed in [5] and [6] is simply stated as follows:

Algorithm A.1 Greedy Hill-Climbing Algorithm

1: Set A $\longleftarrow \phi$
2: **for** $i = 1 \rightarrow k$ **do**
3: Choose a node $v \in V \backslash A$ maximizing $\sigma(A \cup \{v\})$
4: Set A $\longleftarrow (A \cup \{v\})$
5: **end for**

Here k is the initial set of individuals that must be chosen from the online social network graph G. $\sigma()$ represents the influence function. More precisely, given a set S of initial activated nodes (or initial influenced individuals), $\sigma(S)$ represents the final influence spread achieved. $\sigma(S)$ is measured in terms of the expected (or average)

number of finally activated nodes.

This proposed algorithm in [4] guarantees an influence spread with an approximation factor of $(1-1/e)$. Kempe et al. also showed that their algorithm is significantly better than using degree or centrality based heuristics. However, one of the major drawback of this algorithm is its run-time efficiency which is very large even for determining a small set of initial active nodes in a medium sized social network. This is because step 3 in the greedy hill climbing algorithm is run a large number of times for each vertex $v \in V$ to get a more precise value of $\sigma(A \cup \{v\})$.

B. Diffusion Models

There are several diffusion models which describe how the influence(or flow of information etc.) propagates through the online social network. We mention the following two fundamental diffusion models as discussed in [4]:

- **Independent Cascade Model**

  In the independent cascade diffusion model, a *propagation probability* $p_{(u,v)}$ is assigned to each edge $(u, v) \in E$ and an initial set A of nodes is activated. The diffusion process proceeds as follows:

  At a given time step t, any active node u gets one chance to activate all its inactive adjacent neighbors. u can activate $v \in N(u)$ with a probability of $p_{(u,v)}$. Irrespective of whether u succeeds or fails in activating $v \in N(u)$ at time step t, at any further time step, u cannot take part in activating its adjacent nodes. If v has multiple adjacent nodes that are active, each of them will make an attempt to activate their neighbors in an arbitrary order. The nodes activated at time step t will then try to activate their inactive adjacent neighbors at time step $t + 1$. Once no more nodes can be activated, the diffusion process ends.

- **Linear Threshold Model**

  In the *Linear Threshold* diffusion model, each edge $(u, v) \in E$ is assigned a weight $w_{(u,v)}, v \in N(u)$ such that: $\Sigma_{v \in N(u)} w_{(u,v)} \leq 1$. An activation threshold θ_u is selected by each node u, uniformly and randomly from the interval $[0, 1]$ and an initial set A of nodes is activated. The diffusion process proceeds as follows:

  An inactive node u is activated at time step $t+1$ if at time step t: $\Sigma_{v \in N(u)} w_{(u,v)} \geq \theta_u$. Once no more nodes can be activated, the diffusion process ends.

The influence spread of the initial activation set A in both the diffusion models is measured in terms of the final number of nodes that have been activated. Note that in this current research, we consider the independent cascade diffusion model during the experimental analysis.

C. Online Social Network Graph Representation

The online social network is represented as a simple, undirected graph *G = (V,E)*. The set of vertices V is considered to be a union of the following two sets: $V = I \cup CH$, where I is the set of all users in the online social network and CH is the set of available choices to all the users in I. Each user $u \in I$ is assigned a unique integer id between the interval $[0, |I| - 1]$. Each choice $c \in CH$ is also assigned a unique integer id between $[|I|, |I| + |CH| - 1]$.

For any two vertices $u \in I$, $v \in I$, an edge (u, v) exists between u and v only if u and v know each other. An edge (u, c) also exists between $u \in I$ and $c \in CH$ if the user represented by u exhibits a choice c.

D. Key Idea

The central idea of this research originates from a paper by Davis [30] in which he made the following observations:

> [30] The general idea of group effects and opinion leadership can be stated in the language of our theory in terms of a general proposition and a series of subsidiary propositions specifying variations in the process.
>
> **D24.** Person is more likely to adopt an innovation if he has a positive tie of liking or similarity to the innovator.
>
> **D25.** Innovations tend to diffuse rapidly within cliques but slowly between them.
>
> **D26.** Within a group the rate and degree of the acceptance of an innovation is proportional to the degree of liking of the members for each other.
>
> **D27.** Within a group the rate and degree of acceptance of an innovation is proportional to the homogeneity of the members in social characteristic.

The underlying theme of this research is to come up with a new model of online social network graph taking into account the group effects and the degree of similarity between individuals and use this new model for maximizing the diffusion of information across the social network. The idea of clusters of like-minded individuals which form cliques (*critical clusters*) fits well with the observations made in [30]. In the next section, we provide a description of *critical clusters*, their relation to *critical cliques* and the key observations made in [30].

E. Critical Clusters

We attempt to give an overview of the relationship between *critical cliques* and the identification of clusters of like-minded individuals who know each other in an

online social network (termed as *critical clusters*). Consider the properties of such a cluster in an online social network. Primarily, such a cluster should have the following two properties:

- *Everyone knows everyone* - Influencing one individual in such a cluster would lead to a high chance of influencing all other individuals in that cluster. This idea is thus synonymous with the observation made in [30] (D25).

  This property is similar to the property of a critical clique where for any two vertices u and v belonging to the critical clique, there must be an edge between them. The two vertices u and v signify the individuals while the edge between u and v represents the relationship that they are friends or they know each other.

- *Everyone is like-minded* - Two individuals are like-minded if they have the same set of choices. More the similarity or like-mindedness between the individuals, higher the chance of the individuals influencing each other. This idea fits well with the observations made in [30](D24, D26, D27).

  This property is analogous to the property of a critical clique where for any two vertices u and v belonging to the critical clique, u and v have the same closed neighborhood. The closed neighborhood represents the set of choices which are preferred by both the individuals represented by u and v.

Note that the closed neighborhood property of the *critical cliques* induces an additional constraint on the *critical clusters*: For any edge $(u, v), u \in I, v \in I$ in a *critical cluster*, u and v should have the same closed neighborhood w.r.t all the vertices $w \in I$, where w is an adjacent vertex to u and v. Even with this additional constraint, the two properties in a *critical cluster* of *everyone knows everyone* and *everyone is like-minded* are still valid.

F. Calculating Influences Within and Between Critical Clusters

In this section, we define our calculations about how well an individual in the online social network can influence another individual, when they belong to the same or different *critical clusters*. We define this influence by assigning a weight $w \in [0, 1]$ to each edge between two individuals in the online social network graph. This weight represents the *probability of influence* of the edge.

Our calculations for the probability of influence of edges are made on the following observations which are primarily based on the ideas presented in [30]:

1. Within the same *critical cluster*, the influence of one individual over another should be high and uniform throughout the cluster. This is because each *critical cluster* represents a group of like-minded individuals who know each other and have the exact same set of choices. In comparison, influences between individuals in different clusters should be less than the influences within the cluster.

2. As the size of a *critical cluster* increases, the influence within the cluster is likely to decrease as a smaller cluster implies more cohesion between the individuals of the cluster.

3. Apart from the *critical cluster* size, the total choices preferred by each individual in the cluster is also an indication of the like-mindedness among the individuals in that cluster. A higher number of common choices indicates a higher degree of like-mindedness between the individuals and affects the influence within the cluster positively.

Based upon these observations, we define the probability of influence P_C for all internal edges within a *critical cluster* C (comprising of atleast two individuals) as follows:

$$P_C = weight_{cluster} * \frac{\text{min-size-cluster}(G)}{\text{size}(C)} + weight_{choices} * \frac{\text{choices}(C)}{\text{max-choices-in-a-cluster}(G)}$$

where,

min-size-cluster(G) = number of vertices $w \in I$ in the minimum size *critical cluster* (comprising of atleast two individuals) in G,

max-choices-in-a-cluster(G) = number of vertices $c \in CH$ in a *critical cluster* (comprising of atleast two individuals) in G with maximum number of choices,

size(C) = number of vertices $w \in I$ in the *critical cluster* C,

choices(C) = number of vertices $c \in CH$ in the *critical cluster* C

The terms $weight_{cluster}$ and $weight_{choices}$ represent the contribution that comes from the two main factors for defining influences within a *critical cluster*, namely the cluster size and the number of choices of individuals in that cluster. In this research, we assume an equal contribution from both these factors and assign $weight_{cluster} = weight_{choices} = 0.5$. However, for purposes of future research, it would be interesting to study these two factors experimentally and define a measure that evaluates the contribution of these two factors.

Probability of influence of edges between *critical clusters* is assigned a low, constant probability. For any edge (u, v) where $u,v \in I$ and u, v belong to different *critical clusters*, we define this probability as follows:

$P_{u,v} = k * min\{P_C\}$, where,

$k < 1$ is a constant,

$min\{P_C\}$ = minimum probability of influence for internal edges among all the *critical clusters* of atleast two individuals.

Note that in this current research, for experimental purposes, we have assigned a probability of 0.01 for probability of influence of edges between clusters. This constant probability of 0.01 satisfies our defined criteria for probability of influence of an edge

between *critical clusters* in our experiments. In addition to this, it also helps us to perform a comparison of our approach to the greedy approach proposed by Kempe et al. [4] as they also used a constant probability of 0.01 in their experiments.

G. Influence Maximization Algorithm

The Influence Maximization Algorithm comprises of the following steps:

1. **Identify critical clusters**

   Identify the *critical clusters* in the social network graph by using the algorithm of *critical cliques* mentioned in Chapter III. Note that the set of vertices used by the algorithm is $V = I$. Since any vertex $c \in CH$ is in the adjacency list of $v \in I$ if v exhibits a choice of c, the *critical cliques* algorithm would directly give us the *critical clusters*.

2. **Calculate probability of influence of edges**

   Assign the probability of influence for each edge, within and between *critical clusters*.

3. **Determine the initial set of k activation nodes**

   We apply heuristics to determine the selection for an initial set of k individuals. Our heuristics are based on the key observation that within each *critical cluster*, we need to influence only one individual who in turn can strongly influence other individuals in the same *critical cluster*.

   We consider the following heuristics for our initial activation set:

   (a) Selection in the order of *critical cluster* size

   Arrange the *critical clusters* in decreasing order of their sizes and choose an individual $u \in I$ from each of the first k critical clusters.

(b) Selection in the order of number of choices

Arrange the *critical clusters* in decreasing order of their choices count and choose an individual $u \in I$ from each of the first k *critical clusters*.

(c) Selection in the order of probability of influence of internal edges

Arrange the *critical clusters* in decreasing order of the probability of influence of their internal edges and choose an individual $u \in I$ from each of the first k *critical clusters*.

(d) Random selection

Randomly select k individuals after breaking the social network graph into *critical clusters*.

Note that our main proposed heuristics are based on size of the *critical cluster*, choices and the probability of influence of edges within the *critical clusters*. The last heuristic mentioned is for the purpose of comparison of our main heuristics to the *random selection* approach.

4. **Determine the final influence spread**

We take the *independent cascade* model as our diffusion model. In this model, the influence propagates in discrete time steps. In the first time step t, each individual $u \in I$ in the initial activation set(determined in step 3) tries to influence its inactive neighboring individual $v, (u, v) \in E$. It succeeds with a probability of influence $P_{(u,v)}$ which has already been calculated for each edge (u, v) in step 2. An individual u has only one attempt to influence its inactive neighbor v. After this first step, u cannot influence any other of its adjacent neighbors.

The individuals v that were successfully influenced in the first step now try to influence their inactive neighbors w, succeeding with a probability $P_{(v,w)}$.

Just as before, they get one chance to influence their adjacent neighbors. This process continues till there are no more individuals which can be influenced.

At the end of this step, we get the count of the number of individuals activated by the initial activation set. This gives us the measure of the influence spread. Note that we run our algorithm a large number of times (1000 times) to get a better approximation of the influence spread.

H. Runtime Complexity

In this section, we analyse the runtime complexity of our algorithm for determining the initial set of k activation nodes.

In a social network graph, bounding the maximum degree of a user u by a constant is a fairly practical assumption and acceptable as we can (generally) place an upper bound on the number of friends a user u has in an online social network. Also, in the context of this research, the choices u exhibits can be considered to be small when compared to the number of friends $v \in I$ of individual u. Thus, determining the *critical clusters* can be done in $O(|V| + |E|)$ time using the *critical cliques* algorithm discussed in Chapter III.

There can be atmost $|V|$ *critical clusters*. Determining the attributes (size, choices, probability of influence of internal edge) of each *critical cluster* can be done in $O(|E| + |V|)$ time. Finally, we do require a sorting algorithm based on these attributes (in step 3) and since there are atmost $|V|$ *critical clusters*, the sorting algorithm will have a runtime of $O(|V|.log|V|)$. Thus, the overall runtime of our algorithm is $O(|E| + |V|.log|V|)$ for the selection of k initial activation nodes.

I. Experimental Results

1. Collaboration Network Graph

For our experiments, we use the Arxiv HEP-PH collaboration network dataset. This dataset covers scientific collaborations between authors of papers submitted to the category of High Energy Physics - Phenomenology from January 1993 to April 2003.

Using this dataset, we generate an undirected graph for our experiments. The number of nodes in the generated graph is 12008 with a total of 237010 edges. The diameter of the graph is 13 and the clustering coefficient of the graph is 0.6115. Our approach relies on two sets of data nodes: the user nodes and the choice nodes. However, this collaboration network graph cannot be divided into two distinct set of users and their respective choices. Hence, for this experiment, we assume that the set of neighbors of the users in a *critical cluster* is the same as the set of choices for that *critical cluster*. In the context of this dataset, this translates to the fact that the authors which collaborate with each other have the same set of choices(authors) with whom they are collaborating.

In our experiments, the size of our initial set of k activation nodes is taken from the set $S = \{5, 10, 15, 20\}$. For each value of $k \in S$, we run our algorithm a total of 1000 times. After the 1000 iterations for each value of k, we approximate the average number of nodes that are finally influenced for that value of k.

We divide the experiments on the collaboration network graph into following two parts:

1. In the first part, we assign a uniform probability of influence of 1%(.01) to all the edges $(u, v), u, v \in I$. Thereafter, we compare our proposed heuristics based selection approaches to the greedy approach proposed by Kempe et al. [4]

2. In the second part, we calculate the probability of influence as defined in our proposed model. Then, we present the results of our heuristics based selection approaches.

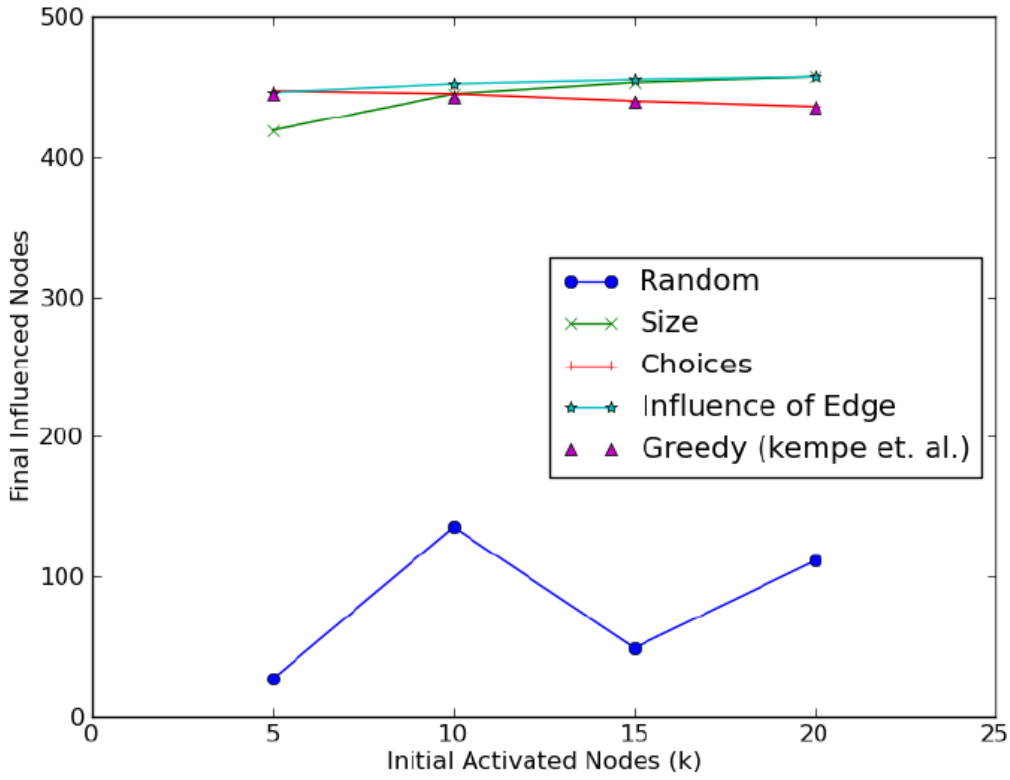

Fig. 4. Comparison of influence maximization heuristics.

The result of the first part of our experiments is shown in the Figure 4. From the experimental results, we observe that the heuristics based on *critical cluster's* sizes and probability of influence of edges are slightly better than the greedy algorithm suggested by Kempe et al. [4] in the independent cascade diffusion model. The heuristic based on the *critical cluster's* choices is comparably equal to the greedy approach. However, we note that randomly selecting nodes after breaking the online social network graph into *critical clusters* performs the worst among all the heuristics.

We show the results of the second part of our experiment on the collaboration

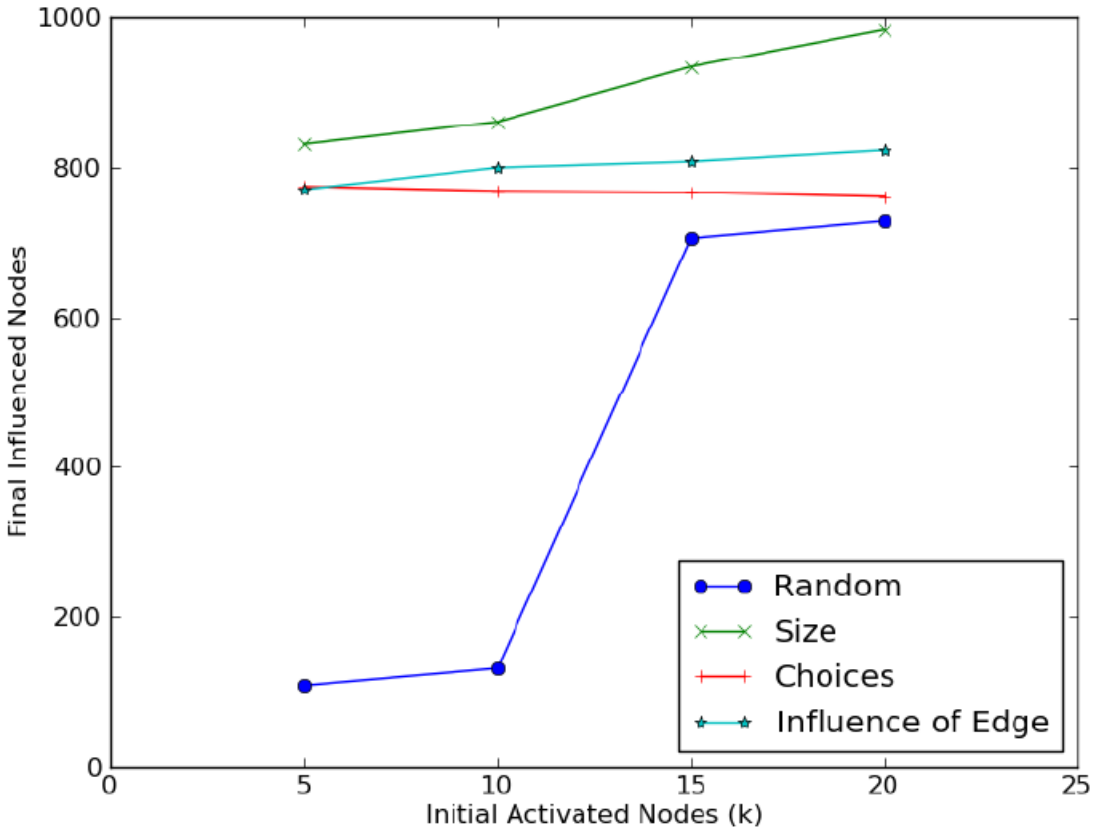

Fig. 5. Comparison of influence maximization heuristics in the proposed model.

network graph in the Figure 5. We note that the heuristics based on *critical cluster's* size, choices and probability of influence of an edge have a better influence spread under our proposed model. The heuristic based on random selection performs the worst in this case.

2. Collaboration Network Graph with Choices

In this set of experiments, we modify the collaboration network graph by adding *choice nodes* which represent the choices of the individuals in the collaboration network. In particular, we perform 3 simulations by adding 2, 5 and 8 choices to the collaboration network graph. For each user u, we flip a coin for each of the choice c and decide with a 50% chance if the user u exhibits the choice c. It is noticeable that as the choices increase, the number of *critical clusters* of $size > 1$ rapidly decreases. This is expected since a set of many choices available to each user means that the chances of a group of users being like-minded are low. For determining the initial set of activation nodes, we run our three main proposed heuristics - based on *critical cluster* size, number of choices in critical cluster and the probability of influence of internal edges in the *critical cluster*.

1. Two Available Choices

   In this simulation, we provide a set of two choices for each user in the collaboration network graph. The results of the performance of the heuristics are shown in Figure 6.

   We notice that since the choices are few, the *critical clusters* ($size > 1$) do not decrease significantly than in the original collaboration graph. Also, the heuristic based on *critical cluster* size performs the best as compared to the other two heuristics.

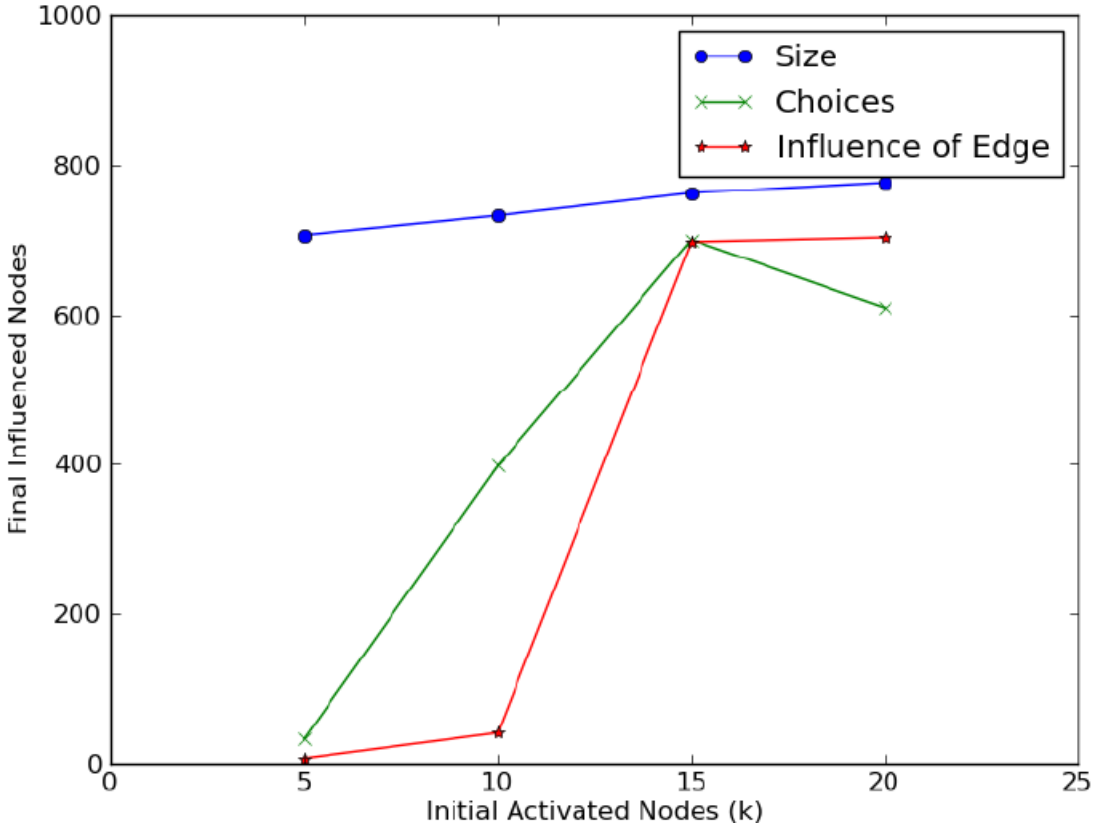

Fig. 6. Comparison of influence maximization heuristics with 2 choices.

2. Five Available Choices

   We provide a total of 5 available choices to each user. The results of the performance of the heuristics are shown in the Figure 7.

   We observe that the maximum influence spread for any value of k is lower than the maximum influence spread for that k from the previous simulation with two choices. This matches our intuition that as the total available choices increases, the similarity between users will be lesser and the influence spread will decrease as a consequence.

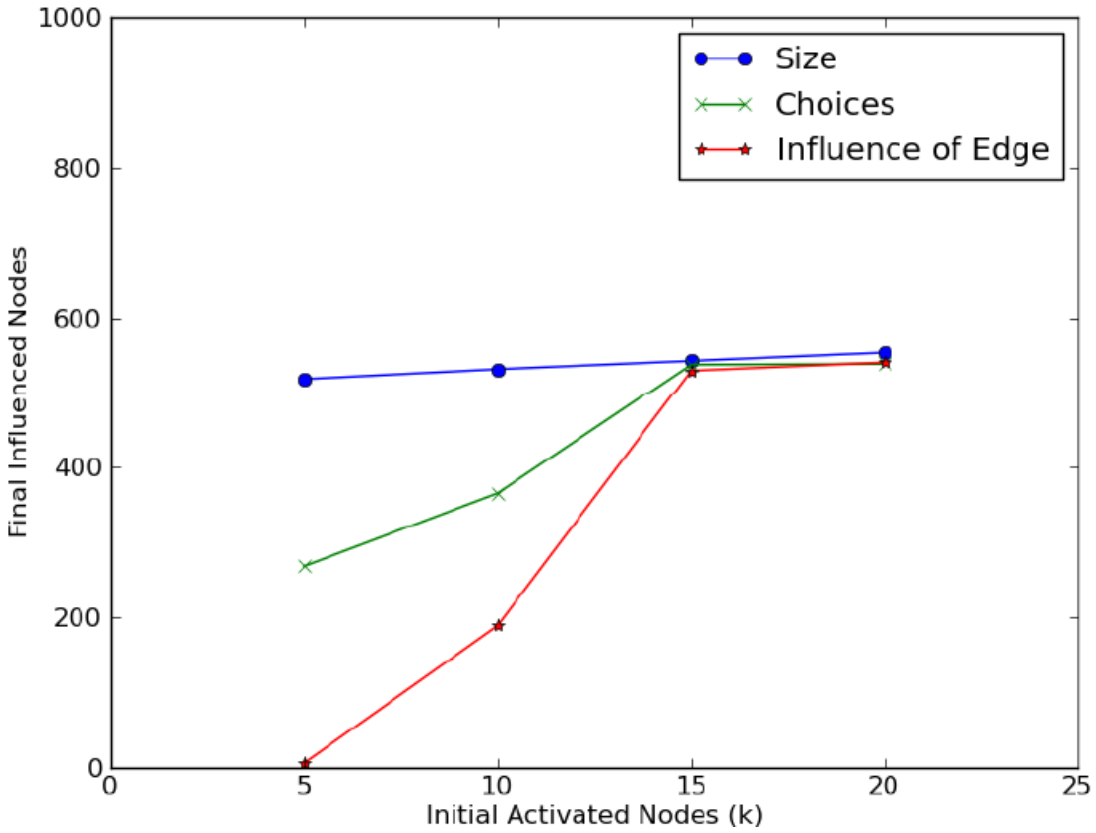

Fig. 7. Comparison of influence maximization heuristics with 5 choices.

3. Eight Available Choices

The setup for this simulation is the same as the previous ones with eight available choices for each user. The results are shown in Figure 8.

We notice that with total available choices as 8, the number of *critical clusters* ($size > 1$) in the social network graph reduce significantly. Also, the maximum influence spread for any k is lower when compared to the maximum influence spread for that k in experiments with total choices as two and five respectively. Since the number of *critical clusters* with $size > 1$ are less and the users are mostly single node *critical clusters*, the influence spread by the various proposed heuristics lie close to each other.

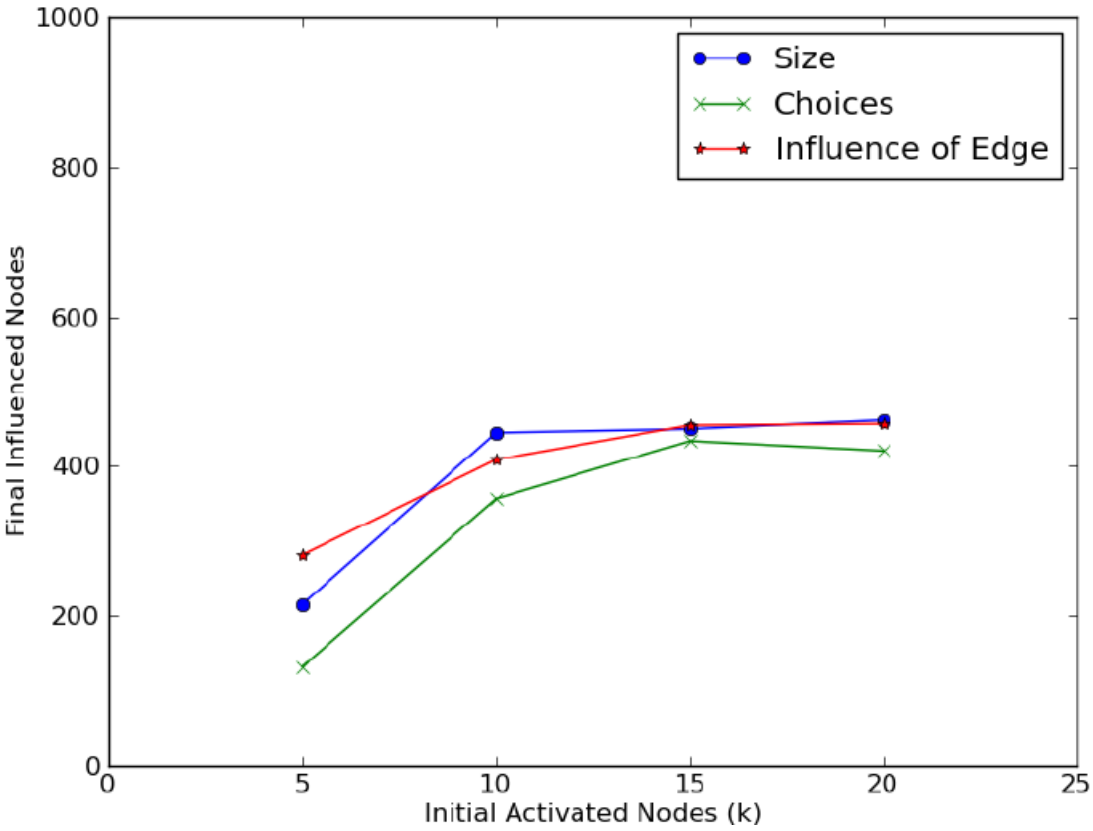

Fig. 8. Comparison of influence maximization heuristics with 8 choices.

3. Critical Cliques in Real World Datasets

We conclude our experimental analysis by analysing real world datasets for *critical cliques*. The motivation for this analysis is to show that the proposed model of online social network in this research has a practical application to real world data.

Table I. Critical cliques in real world datasets.

Description	**Nodes**	**Edges**	**CC Nodes**	**CC**
Collaboration network of Arxiv Astro Physics	18,772	396,160	5,215	1770
Collaboration network of Arxiv Condensed Matter	23,133	186,936	7,813	3059
Collaboration network of Arxiv General Relativity	5,242	28,980	1,847	712
Collaboration network of Arxiv High Energy Physics	12,008	237,010	3,558	1112
Collaboration network of Arxiv High Energy Physics Theory	9,877	51,971	2,222	955
Email communication network from Enron	36,692	367,662	8,704	3592

Table I lists our analysis on the real world datasets. The datasets are taken

from *Stanford Large Network Dataset Collection*. Our analysis is based on the real world undirected graphs present in this collection. For each dataset, we calculate the number of *critical cliques* comprising of atleast two nodes (column *CC* in Table I) and the total number of such *critical cliques* nodes (column *CC Nodes* in Table I).

Our analysis shows that we do get a sufficient number of *critical cliques* of $size > 1$ (and *critical cliques* nodes) in the real world data. Based on these findings, we believe that our online social network modeled on *critical clusters* has practical applications to real world data.

CHAPTER V

FUTURE WORK AND CONCLUSION

A. Future Work

In this research, we presented an algorithm for breaking a simple, undirected graph into *critical cliques* and later on, we presented an application of the *critical cliques* in the domain of social computing. We presented a new model of online social network by partitioning the online social network graph into *critical clusters* and utilized their structural properties to solve the problem of influence maximization. However, we believe that the following key points related to our work in the influence maximization problem have further scope of future work:

1. Relaxing the Criteria of Membership in Critical Clusters

Our definition of *critical clusters* is restrictive in the sense that it requires the set of people within a *critical cluster* to have the exact same set of choices. In addition to this, the inherited property of *critical cliques* also induces a constraint that the individuals belonging to a *critical cluster* must have the same closed neighborhood (in terms of the individuals known) as all the other individuals in that cluster.

We believe that we can relax the definition of *critical clusters* by allowing the following:

- Allowing individuals belonging to a *critical cluster* to have a majority of their choices to be the same
- Relaxing the property of *critical clusters* of being *cliques*. In other words, we could allow *critical cluster* to be a dense subgraph of the online social network rather than a clique.

Relaxing the criteria for membership of *critical clusters* would help us in getting a better realistic partitioning of the online social network graph. Thus, this forms an interesting area of future research which we would like to explore further.

2. Determining Probability of Influence in Critical Clusters

We consider two factors in determining the influence spread within a *critical cluster* C which are the size of C and the number of choices in that cluster C. An equal contribution from both these factors is assumed while determining the probability of influence of an edge in a *critical cluster*. However, we believe that the contribution from both these factors can be more effectively determined by analysing real world data sets and studying how the influence spread varies with changes in the size of a cluster and the number of choices within a cluster.

B. Conclusion

The main contribution of this research can be summarized as follows:

- Even though there are a number of linear time modular decomposition algorithms for undirected graphs that can be used to obtain *critical cliques*, yet using the modular decomposition process to obtain *critical cliques* is not a preferred approach because of the following reasons:
  (a) It is complex to implement the linear time modular decomposition algorithms.
  (b) The use of intermediate data structures like PQ trees, ordered chain partitions etc. add to the overall complexity (constant factors).

In this research, we proposed a simple algorithm to directly decompose a graph

into its constituent *critical cliques*. Our algorithm utilized the fundamental property of a *critical clique* which is that it is a maximal clique under the condition that all the vertices within this clique have the same closed neighborhood. The main advantage of our algorithm for *critical cliques* is that it is simpler and easier to implement. Also, the constant factors associated with the runtime complexity of our algorithm are small as discussed earlier in Chapter III, Section D.

- We presented a new application of *critical cliques* in the domain of social network analysis. We proposed a new model for structuring the online social network based on the idea of *critical cliques*. Also, we proposed a set of heuristics based on the properties of *critical clusters* for identifying the initial set of nodes in an online social network graph to solve the problem of influence maximization. The key contributions of our work related to influence maximization in online social networks are:

    - A proposed new model of online social network in terms of *critical clusters* which takes into account the structure of the online social network graph.
    - A new and faster algorithm for selection of initial set of activation nodes based on our proposed model. The greedy algorithm proposed by Kempe et al. in [4] has a running time of $O(k.|V|.R.|E|)$ to select k initial activation nodes. Here, R is the number of times the influence spread is determined for each $v \in V$ to get a better approximation for the influence spread of v. In comparison, our algorithm has a runtime complexity of $O(|E| + |V|.log|V|)$ for determining the initial set of k activation nodes.

REFERENCES

[1] J. Guo, "Fixed-parameter algorithms for graph-modeled date clustering," in *Proc. of the 6th Annual Conference on Theory and Applications of Models of Computation (TAMC)*, 2009, pp. 39–48.

[2] J. Guo, "A more effective linear kernelization for cluster editing," *Theoretical Computer Science*, vol. 410, no. 8-10, pp. 718–726, 2009.

[3] J. Chen and J. Meng, "A 2k kernel for the cluster editing problem," in *Proc. of the 16th Annual International Conference on Computing and Combinatorics (COCOON)*, 2010, pp. 459–468.

[4] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 137–146.

[5] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 199–208.

[6] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large scale social networks," in *Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 1029–1038.

[7] T. Gallai, "Transitiv orientierbare graphen," *Acta Mathematica Hungarica*, vol. 18, no. 1-2, pp. 25–66, 1967.

[8] Rolf H. Möhring, "Algorithmic aspects of comparability graphs and interval graphs," *Graphs and Orders*, pp. 41–101, 1985.

[9] D.G. Corneil, Y. Perl, and L.K. Stewart, "A linear recognition algorithm for cographs," *SIAM Journal of Computing*, vol. 14, pp. 926–934, 1985.

[10] A. Pnueli, S. Even, and A. Lempel, "Transitive orientation of graphs and identication of permutation graphs," *Canadian Journal of Mathematics*, vol. 23, pp. 160–175, 1971.

[11] S. Böcker, Quang Bao Anh Bui, and Anke Truss, "An improved fixed-parameter algorithm for minimum-flip consensus trees," in *Proc. of the 3rd International Conference on Parameterized and Exact Computation (IWPEC)*, 2008, pp. 43–54.

[12] F. Protti and J.L. Szwarcfiter M. Dantas da Silva, "Applying modular decomposition to parameterized cluster editing problems," *Theory of Computing Systems*, vol. 44, no. 1, pp. 91–104, 2009.

[13] D.D. Cowan, L.O. James, and R.G. Stanton, "Graph decomposition for undirected graphs," in *3rd S-E Conference on Combinatorics, Graph Theory and Computing*, 1972, pp. 281–290.

[14] W.H. Cunningham, "Decomposition of directed graphs," *SIAM Journal on Algebraic and Discrete Methods*, vol. 3, pp. 214–228, 1982.

[15] M. Golumbic, "Comparability graphs and a new matroid," *Journal of Combinatorial Theory*, vol. 22, pp. 68–90, 1977.

[16] M. Habib and M.C. Maurer, "On the X-join decomposition for undirected graphs," *Discrete Applied Mathematics*, vol. 1, no. 3, pp. 201–207, 1979.

[17] G. Steiner, "Machine scheduling with precedence constraints," Ph.D. dissertation, Department of Combinatorics and Optimization, University of Waterloo,

Waterloo, Ontario, 1982.

[18] J.H. Muller and J. Spinrad, "Incremental modular decomposition," *Journal of the ACM*, vol. 36, no. 1, pp. 1–19, 1989.

[19] A. Ehrenfeucht, H.N. Gabow, R.M. McConnell, and S.J. Sullivan, "An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of 2-structures and modular decomposition of graphs," *Journal of Algorithms*, vol. 16, pp. 283–294, 1994.

[20] R.M. McConnell and J. Spinrad, "Linear-time modular decomposition and efficient transitive orientation of comparability graphs," in *5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1994, pp. 536–545.

[21] A. Cournier and M. Habib, "A new linear algorithm of modular decomposition," in *Proc. of the 19th International Colloquium on Trees in Algebra and Programming (CAAP)*, 1994, pp. 68–84.

[22] R.M. McConnell and J.P. Spinrad, "Modular decomposition and transitive orientation," *Discrete Mathematics*, vol. 201, pp. 189–241, 1999.

[23] M. Tedder, D. Corneil, M. Habib, and C. Paul, "Simpler linear time modular decomposition via recursive factorizing permutations," in *International Colloquium on Automata, Languages and Programming (ICALP)*, 2008, pp. 634–645.

[24] E. Dahlhaus, J. Gustedt, and R.M. McConnell, "Efficient and practical algorithm for sequential modular decomposition algorithm," *Journal of Algorithms*, vol. 41, no. 2, pp. 360–387, 2001.

[25] R.M. McConnell and J.P. Spinrad, "Ordered vertex partitioning," *Discrete Mathematics and Theoretical Computer Science*, vol. 4, pp. 45–60, 2000.

[26] M. Habib, C. Paul, and L. Viennot, "Partition refinement : an interesting algorithmic tool kit," *International Journal of Foundation of Computer Science*, vol. 10, no. 2, pp. 147–170, 1999.

[27] Michel Habib and Christophe Paul, "A survey on algorithmic aspects of modular decomposition," *Computer Science Review*, vol. 4, no. 1, pp. 41–59, 2010.

[28] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007, pp. 420–429.

[29] M. Kimura and K. Saito, "Tractable models for information diffusion in social networks," in *Proc. of the 10th European conference on Principle and Practice of Knowledge Discovery in Databases (PKDD)*, 2006, pp. 259–271.

[30] J. A. Davis, "Structural balance, mechanical solidarity and interpersonal relations," *American Journal of Sociology*, vol. 68, no. 4, pp. 444–462, 1963.

[31] F. J. Radermacher Rolf H. Möhring, "Substitution decomposition for discrete structures and connections with combinatorial optimization," *Annals of Discrete Mathematics*, vol. 19, pp. 257–356, 1984.

VITA

Name Nikhil Pandey

Address Department of Computer Science,
301 Harvey R. Bright Building,
College Station, TX 77843-3112

Email Address nikhil@cse.tamu.edu

Education

- Master of Science, Computer Science, May 2012, Texas A&M University, College Station, TX
- Bachelor of Technology, Computer Science and Engineering, May 2007, Motilal Nehru National Institute of Technology, Allahabad, U.P, India