MapReduce-Accelerated Framework for Identifying Minimum-Sized Influential Vertices on Large-Scale Weighted Graphs

Ying Xie\textsuperscript{1}, Jing (Selena) He\textsuperscript{1}, and Vijay V. Raghavan\textsuperscript{2}
\textsuperscript{1}Department of Computer Science, Kennesaw State University, Kennesaw, GA, USA
\textsuperscript{2}The Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA, USA

Abstract—Weighted graphs can be used to model any data sets composed of entities and relationships. Social networks, concept networks and document networks are among the types of data that can be abstracted as weighted graphs. Identifying Minimum-sized Influential Vertices (MIV) in a weighted graph is an important task in graph mining that gains valuable commercial applications. Although different algorithms for this task have been proposed, it remains challenging for processing web-scale weighted graph. In this paper, we propose a highly scalable algorithm for identifying MIV on large-scale weighted graph using the MapReduce framework. The proposed algorithm starts with identifying an individual zone for every vertex in the graph using an $\alpha$-cut fuzzy set. This approximation allows us to divide the whole graph into multiple subgraphs that can be processed independently. Then, for each subgraph, a MapReduce based greedy algorithm is designed to identify the minimum-sized influential vertices for the whole graph.

Keywords: MapReduce Framework, Minimum-sized Influential Vertices, Large-Scale Weighed Graph, Big Data Analysis.

1. Introduction

Weighted graphs can be used to model any data sets composed of entities and relationships. Social networks, concept networks, and document networks are among the types of data that can be abstracted as weighted graphs. Identifying Minimum-sized Influential Vertices (MIV) in a weighted graph is an important task in graph mining that gains valuable commercial applications. Consider the following hypothetical scenario as a motivating example. A small company develops a new online application and wants the application and start influencing their friends on the social networks to use it. And their friends would influence their friends’ friends and so on, and thus through the word-of-mouth effect a large population in the social network would adopt this application. In sum, the MIV problem is whom to select as the initial users (keep the size as small as possible or under some budget) so that they eventually influence the largest number of people in the network.

The problem is first introduced for social networks by Domingos and Richardson in [2] and [3]. Subsequently, Kempe et al. [4] proved this problem to be NP-hard and propose a basic greedy algorithm that provides good approximation to the optimal solution. However, the greedy algorithm is seriously limited in efficiency because it needs to run Monte-Carlo simulation for considerably long time period to guarantee an accurate estimate. Although a number of successive efforts have been made to improve the efficiency, the state-of-the-art approaches still suffer from excessively long execution time due to the high-computational complexity for large-scale weighted graph. Furthermore, the graph structure of real-world social networks are highly irregular, making MapReduce acceleration a non-trivial task. For example, Barack Obama, the U.S. president, has more than 11 million followers in Twitter, while more than 90% of Twitter users, the follower number is under 100 [8]. Such irregularities may lead to severe performance degradation.

On the other hand, MapReduce framework has recently been widely used as a popular general-purpose computing framework and has also been shown promising potential in accelerating computation of graph problems such as breadth first searching and minimum spanning tree [5], [6], [7], due to its parallel processing capacity and ample memory bandwidth. Therefore, in this paper, we explore the use of MapReduce framework to accelerate the computation of MIV in large-scale weighted graphs.

The proposed framework starts with identifying an individual zone for every vertex in the graph. The individual zone of a given vertex is the set of vertices that the given vertex can influence. To design a scalable algorithm to address this, we approximate individual zone by using the concept of $\alpha$-cut fuzzy set. This approximation allows us to reduce the complexity of multi-hop influence propagation to the level of single-hop propagation. Subsequently, we aim to find a minimum-sized set of vertices whose influence (the
formal definition will be presented in Section 3) reaches a pre-defined threshold. To reach this goal, a MapReduce-based greedy algorithm is designed by processing individual zones (the formal definition will be presented in Section 3) for all vertices.

As a summary, the contribution of this paper can be summarized as follows:

- A fuzzy propagation model was proposed to describe multi-hop influence propagation along social links in weighted social networks;
- An $\alpha$-cut fuzzy set called individual zone was defined to approximate multi-hop influence propagation from each vertex.
- MapReduce algorithms were designed to locate each individual zone and then identify Minimum-sized Influential Vertices (MIV) using a greedy strategy.

This remainder of this paper is organized as follows: Section 2 reviews related literatures of the MIV problem. Network model and the formal definition of the MIV problem are given in Section 3. The MapReduce-accelerated framework are presented in Section 4 and Section 5. Finally, the work is concluded in Section 6.

2. Related Work

Finding the influential vertices and then eventually influencing most of the population in the network is first proposed by Domingos et al. in [2], [3]. They model the interaction of users as a Markov random field and provide heuristics to choose users who have large influence in network. Kempe et al. [4] formulate the problem as a discrete optimization problem and propose a greedy algorithm. However, the greedy algorithm is time-consuming. Hence, recently huge amount of researchers try to improve the greedy algorithm in two ways. One is reduce the number of individual searched in the graph. The other is improving the efficiency of calculating the influence of each individual. Leskovec et al. [9] propose an improved approach which is called CELF to reduce the number of individual searched in the graph. Later, Goyal et al. [10] propose an extension to CELF called CELF++, which can further reduce the number. Kimura et al. [11] utilize the Strong Connected Component (SCC) to improve the efficiency of the greedy algorithm.

Although many algorithms are proposed to improve the greedy algorithm, they are not efficient enough for the large scale of current social networks. Hence, some works are proposed to fit for large-scale networks. Chen et al. proposed a method called MixGreedy [12] that reduces the computational complexity by computing the marginal influence spread for each node and then selects the nodes that offers the maximum influence spread. Subsequently, Chen et al. [13] use local arborescence of the most probable influence path between two individual to further improve the efficiency of the algorithm. However, both of the algorithms provide no accuracy guarantee. In [14], Liu et al. propose ESMCE, a power-law exponent supervised Monte-Carlo method that efficiently estimates the influence spread by randomly sampling only a portion of the nodes. There have been also many other algorithm and heuristics proposed for improving the efficiency issues for large-scale social networks, such as [15], [16]. However, all of the aforementioned improvements are not effective enough to reduce execution time to an acceptable range especially for large-scale networks.

Completely different from the previous mentioned work, Liu et al. [17] present a GPU framework to accelerate influence maximization in large-scale social networks called IMGPU, which leveraging the parallel processing capability of Graphics Processing Unit (GPU). The authors first design a bottom-up traversal algorithm with GPU implementation to improve the existing greedy algorithm. To best fit the bottom-up algorithm with the GPU architecture, the authors further develop an adaptive $K$-level combination method to maximize the parallelism and reorganize the influence graph to minimize the potential divergence. Comprehensive experiments with both real-world and synthetic social network traces demonstrate that the propose IMGPU framework outperform the state-of-the-art influence maximization algorithm up to a factor of 60.

In this paper, we focus on addressing the MIV problem on large-scale weighted graphs using the MapReduce framework. The proposed method first improve the algorithm efficiency by divide the whole graph into some subgraphs using fuzzy propagation model. Subsequently, a MapReduce greedy algorithm is presented to search the best candidates in each subgraphs to achieve highly parallelism. The proposed framework show potential to scale up to extraordinarily large-scale graphs.

3. Graph Model and Problem Definition

3.1 Graph Model

We model a weighted graph by an undirected graph $G(V, E, W(E))$, where $V$ is the set of $N$ vertices, denoted by $v_i$, and $0 \leq i < N$. $i$ is called the vertex ID of $v_i$. An undirected edge $e_{ij} = (v_i, v_j) \in E$ represents weights between the pair of vertices. $W(E) = \{p_{ij} \mid (v_i, v_j) \in E, 0 < p_{ij} \leq 1, \text{ else } p_{ij} = 0\}$, where $p_{ij}$ indicates the weights between vertices $v_i$ and $v_j$. For simplicity, we assume the links are undirected (bidirectional), which means two linked vertices have the same weight ($i.e., p_{ij}$ value) on each other. Figure 1 shows an example of a weighted graph.

3.2 Problem Definition

The objective of the MIV problem is to identify a subset of influential vertices in the weighted graph. Such that, eventually large number of vertices in the graph can be influenced by these initially selected vertices. As we mentioned in Section 1, we first partition the whole graph
Algorithm 1: Mapper Part

Method Map (vertexID id, vertexRecord: r)

Instantiate Vertex v from id and r;

\( i = 0, j = 0, k = 0; \)

\textbf{for} \( i < v:\text{ColorToTargets}.\text{size}() \) \textbf{do}

\textbf{if} \( v:\text{ColorToTargets}[i] = G \) \textbf{then}

\textbf{for} \( j < v:\text{NeighborsID}.\text{size}() \) \textbf{do}

\textbf{if} \( \text{membershipToTarget} > \alpha \) \textbf{then}

Instantiate Vertex vv for \( \text{Neighbors}[j] \);

\( vv:\text{ID} = \text{NeighborsID}[j]; \)

\( vv:\text{NeighborsID} = \text{Null}; \)

\( vv:\text{MembershipOfNeighbors} = \text{Null}; \)

\textbf{for} \( k < v:\text{ColorToTargets}.\text{size}() \) \textbf{do}

\textbf{if} \( i == k \) \textbf{then}

\( vv:\text{MembershipToTargets}[k] = \text{membershipToTarget}; \)

\( vv:\text{ColorToTargets}[k] = G; \)

\( vv:\text{parentToTargets}[k] = v:\text{VertexID}; \)

\textbf{else}

\( vv:\text{MembershipToTargets}[k] = 0; \)

\( vv:\text{ColorToTargets}[k] = W; \)

\( vv:\text{ColorToTargets}[k] = -1; \)

Create record \( rr \) from vv in the following format:

- list of adjacent vertices and weights
- membership values target vertices
- color decorations towards target vertices
- immediate parent towards target vertices

\text{EMIT}(vv:\text{VertexID}, rr); \)

\( v:\text{ColorToTargets}[i] = B; \)

Create record \( r \) from \( v \) in the following format:

- list of adjacent vertices and weights
- membership values target vertices
- color decorations towards target vertices
- immediate parent towards target vertices

\text{EMIT}(id, r);
Algorithm 1: Reducer Part

Method Reduce(vertexID id, \{r1, r2, r3, \cdots, rl\})

Instantiate Vertex v from id;
v.VertexID = id;
v.NeighborsID = Null;
v.MembersOfNeighbors = \{0, 0, 0, \cdots\};
v.ColorToTargets = \{W, W, W, \cdots\};
v.ParentToTargets = \{-1, -1, -1, \cdots\};
i = 0, j = 0;
for each \(r_i\) in \{r1, r2, r3, \cdots, rl\} do
    Instantiate Vertex \(v\) from \(id\) and \(r_i\):
    if \(v.NeighborsID! = Null\) then
        v.NeighborsID = vv.NeighborsID;
        v.MembersOfNeighbors = vv.MembershipOfNeighbors;
    for \(j < vv.ColorToTargets.size()\) do
        if \(vv.ColorToTargets[j] > v.ColorToTargets[j]\) then
            v.ColorToTargets[j] = vv.ColorToTargets[j];
    for \(j < vv.MembershipToTargets.size()\) do
        if \(vv.MembershipToTargets[j] > v.MembershipToTargets[j]\) then
            v.MembershipToTargets[j] = vv.MembershipToTargets[j];
    for \(i < v.ColorToTargets.size()\) do
        if \(v.ColorToTargets[i] = G\) then
            Increment a predefined job counter called numberOfIterations;

Create record \(r\) from \(v\) in the following format:
list of adjacent vertices and weights | membership values target vertices | color decorations towards target vertices |

Immediate parent towards target vertices;
EMIT(id, r);

into subgraphs by applying a fuzzy propagation modes.
The subgraph is called individual zone for each vertex.
Subsequent, we formally define individual zone as follows:

Definition 3.1: individual zone (Zone\(_v\)). For weighted graph \(G(V, E, W(E))\), the individual zone is a fuzzy set \((U_v, M_v)\), where, \(U_v\) is the set of vertices, and \(M_v\) is a function: \(U_v \rightarrow [0, 1]\), such that for any vertex \(v \in U_v\), we have

\[
M_v(x) = \begin{cases} 
1, & \text{if } x = v; \\
\prod_{e_{ij} \in P_{xv}} W(e_{ij}), & \text{otherwise.}
\end{cases}
\]

where \(P_{xv}\) is the path from vertex \(x\) to vertex \(v\), \(e_{ij}\) is an edge in the path. We further define \(W(e_{ii}) = 1\).

Another important terminology influence must be formally defined before the problem definition.

Definition 3.2: influence (\(\zeta\)). For weighted graph \(G(V, E, W(E))\), the influence of vertex \(v\) is denoted by \(\zeta_v\), which is the sum of the membership value of all vertices in Zone\(_v\).

Now, we are ready to define the Minimum-sized Influential Vertices (MIV) problem as follows:

Definition 3.3: Minimum-sized Influential Vertices (MIV). For weighted graph \(G(V, E, W(E))\), the MIV problem is to find a minimum-sized influential vertices \(x \subseteq V\), such that \(\forall v \in x\), \(\zeta_v \geq N \times x\%\), where \(x\%\) is a pre-defined threshold.

4. MapReduce Algorithm for Identifying Individual Zones

To scale the MIV problem to a large-scale weighted graph, we approximate individual zones by using \(\alpha\)-cut fuzzy sets. That is, given a vertex \(v\), the \(\alpha\)-cut individual zone of \(v\) contains and only contains all vertices whose membership value towards \(v\) is greater than or equal to the give parameter \(\alpha\). For simplicity, in the description of the MapReduce algorithms shown in Section 4 and 5, individual zone actually means \(\alpha\)-cut individual zone.

A given weighted graph will be represented by using adjacency lists, a similar representation used in MapReduce based algorithms for breath first searching and minimum spanning tree [5], [6], [7]. For instance, the weight graph shown in Figure 1 is described as follows:

1) 2(0.7), 7(0.6)
2) 1(0.7), 3(0.8), 6(0.6), 8(0.8)
3) 2(0.8), 4(0.7), 5(0.8), 8(0.6), 6(0.9)
4) 3(0.7), 5(0.5), 9(0.6)
5) 3(0.8), 4(0.7), 8(0.7)
6) 2(0.6), 3(0.9)
7) 1(0.6), 8(0.6)
8) 2(0.8), 3(0.6), 5(0.7), 7(0.6), 9(0.7)
9) 8(0.7), 4(0.6)

For a large-scale weighted graph, we divide its adjacency lists into \(k\) equal-sized files, where \(k = total/size/block\).
size (total – size is the total size of the adjacency lists of the graph data, and block – size is the block size of the HDFS of the Hadoop cluster). Assume there are \( n \) vertices for each of the \( k \) files. Then we select \( m \) vertices from each file respectively to form a set of target vertices for which we identify individual zones. In other words, the execution of the following MapReduce algorithm is able to identify individual zones for \( k \times m \) target vertices. Therefore, we just need to run \( n/m \) times of this algorithm in order to identify individual zones for all vertices. Fortunately, the \( n/m \) times of executing this algorithm is totally independent to each other, thus can run in a completely parallel manner.

As an illustration, we assume \( k = 2 \) and \( n = 4 \) for the weighted graph given in Figure 1. Then, let \( m = 2 \), i.e., for each of the 2 files, we select 2 vertices as the target vertices. Assume for the first run of the algorithm, we select vertices 1 and 2 from File 1, and vertices 5 and 6 from File 2. Then we will have the following two input files to identify individual zones for vertices 1, 2, 5, and 6.

**File 1:**
1. 2(0.7), 7(0.6) | 1, 0, 0, 0 | G, W, W, W | 0, -1, -1, -1
2. 10(0.7), 3(0.8), 6(0.6), 8(0.8) | 0, 1, 0, 0 | W, G, W, W | 1, 0, -1, -1
3. 2(0.8), 4(0.7), 5(0.8), 8(0.6) | 0, 0, 0, 0 | W, W, W, W | 1, -1, -1, -1
4. 3(0.7), 5(0.5), 9(0.6) | 0, 0, 0, 0 | W, W, W, W | 1, -1, -1, -1

**File 2:**
5. 3(0.8), 4(0.7), 8(0.7) | 0, 0, 1, 0 | W, W, G, W | -1, 0, -1, -1
6. 2(0.6), 3(0.9) | 0, 0, 0, 1 | W, W, W, G | -1, -1, -1, 0
7. 1(0.6), 8(0.6) | 0, 0, 0, 0 | W, W, W, W | -1, -1, -1, -1
8. 7(0.6), 2(0.8), 3(0.6), 5(0.7), 9(0.7) | 0, 0, 0, 0 | W, W, W, W | -1, 0, -1, -1
9. 8(0.7), 4(0.6) | 0, 0, 0, 0 | W, W, W, W | -1, -1, -1, -1

In the two input files shown above, each vertex is represented in the following format: VertexID list of adjacent vertices and weights | membership values target vertices | color decorations towards target vertices | immediate parent towards target vertices.

Taking vertex 1 as an example, its membership values to the four target vertices (1, 2, 5, and 6) are initialized to be 1, 0, 0, 0, respectively. Its color decoration is set to be Gray (G), White (W), White (W), White (W) respectively, where the first G means that more vertices belonging to the individual zone of the first target vertex need to be located starting from this vertex; the rest W means that, for other target vertices, no immediate action is needed from this vertex; the other possible color value is Black (B), which means no further development from the current vertex is needed for the corresponding target vertex. We further assume the ordinal among the three color values are \( B > G > W \). Its immediate parent vertices towards each of the target vertices are initialized to be 0, -1, -1, and -1, respectively, where the first 0 means that this vertex itself is the first target vertex; and the rest -1 mean that its parent vertex to the rest of the target vertices remains unknown for right now.

We further use the following data structure Vertex to hold the information on each individual vertex.

- **ID:** int
- **NeighborsID:** List < Integer >
- **MembershipOfNeighbors:** List < Double >
- **MembershipToTargets:** Array < Double >
- **ColorToTargets:** Array < Char >
- **ParentToTargets:** Array < Integer >

Then, the MapReduce Algorithm can be described in Algorithm 1.

When this MapReduce job is executed on input File 1 and File 2, it will generate the following output, if \( \alpha = 0.5 \):
1. 2(0.7), 7(0.6) | 1, 0.7, 0, 0 | B, G, G, W | 0, 2, -1, -1
2. 10(0.7), 3(0.8), 6(0.6), 8(0.8) | 0.7, 1, 0, 0.6 | G, B, W, G | 1, 0, -1, 6
3. 2(0.8), 4(0.7), 5(0.8), 8(0.6), 6(0.9) | 0, 0.8, 0.8, 0.9 | W, B, G, W | 1, 0, -1, 2, 5
4. 3(0.7), 5(0.5), 9(0.6) | 0, 0, 0, 0.1 | W, W, G, W | -1, -1, 5, -1
5. 3(0.8), 4(0.7), 8(0.7) | 0, 0, 0, 0.1 | W, W, W, G | 1, -1, -1, -1
6. 2(0.6), 3(0.9) | 0, 0, 0, 1 | W, W, W | -1, -1, -1, 0
7. 1(0.6), 8(0.6) | 0, 0, 0, 0 | W, W, W | -1, -1, -1, -1
8. 7(0.6), 2(0.8), 3(0.6), 5(0.7), 9(0.7) | 0.8, 0.7, 0.7 | W, G, G, W | 1, 0, 2, 5
9. 8(0.7), 4(0.6) | 0, 0, 0, 0 | W, W, W | -1, -1, -1, -1

Since the output records contains \( G \) color, the job counter numberOfIterations will be greater than 0. So we run above MapReduce job for another iteration by using the output of the first run as input. This process will continue until no record in output contains any \( G \) color. Then the output contains information on individual zones for the target vertices 1, 2, 5, and 6. In the same way, we are able to obtain individual zones for vertices 3, 4, 7, 8, and 9.

### 5. MapReduce Algorithm for solving MIV

Using the graph shown in Figure 1 as an illustration again, the output of Algorithm on this graph can be easily converted to the following format by a MapReduce process. Each record represents the \( \alpha \)-cut individual zone for a vertex. Given a vertex, its influence is the sum of all the membership values in its \( \alpha \)-cut individual zone. For example, the influence of vertex 1 is \( 0.7 + 0.6 + 0.56 + 0.56 = 2.42 \).

Now, the task is to find a minimum-sized influential vertices whose influence reaches \( N \times x\% \).

1. 2(0.7), 7(0.6), 8(0.56), 3(0.56)
2. 1(0.7), 8(0.8), 6(0.6), 3(0.8), 4(0.56), 5(0.64), 9(0.56)
3. 2(0.8), 6(0.9), 8(0.64), 5(0.8), 4(0.7), 10(0.56)
4. 3(0.7), 5(0.5), 9(0.6), 2(0.56), 6(0.63)
5. 8(0.7), 3(0.8), 4(0.56), 2(0.64), 6(0.72)
6. 2(0.6), 3(0.9), 8(0.54), 5(0.72), 4(0.63)
7. 1(0.6), 8(0.6)
8. 7(0.6), 2(0.8), 3(0.6), 5(0.7), 9(0.7), 1(0.56), 6(0.54)
9. 8(0.7), 4(0.6)

We design a MapReduce based greedy algorithm for computing this task. Let the minimum-sized set of influential vertices be denoted as \( S \). We also introduce another set denoted as \( I \), which includes all the vertices that are influenced by vertices in \( S \) as well as their maximum membership values towards all influential vertices in \( S \). For instance, for the graph shown in Figure 1, if \( S = \{2, 8\} \), then \( I = \{2(1), 8(1), 1(0.7), 6(0.6), 3(0.8), 4(0.56), 5(0.7), 9(0.7), 7(0.6)\} \), and the influence of \( S \) is \( 1 + 1 + 0.7 + 0.6 + 0.8 + 0.56 + 0.7 + 0.7 + 0.6 = 6.66 \).
A MapReduce based greedy algorithm for identifying minimum-sized influential vertices can be described in Algorithm 2:

\section*{Algorithm 2: Mapper Part}
\begin{algorithm}
\textbf{Method Map} (vertexID id, vertexRecord: r)  
\begin{itemize}
    \item \texttt{I}_0 = \text{I};  
    \item \text{if} id \text{ is in } \text{I} \text{ then} 
        \begin{itemize}
            \item reset its membership value in \text{I} to be \text{1};  
            \item \text{add} id(1) \text{ to } \text{I};
        \end{itemize}
    \item \text{for each vertex } v_i \text{ in } r \text{ do}
        \begin{itemize}
            \item \text{if} \text{ v}_i \text{ is in set } \text{I} \text{ then}
                \begin{itemize}
                    \item set its membership value recorded in \text{I} \text{ be } \text{mm};
                    \item if \text{m} > \text{mm} \text{ then}
                        \begin{itemize}
                            \item replace \text{mm} \text{ with } \text{m} \text{ in } \text{I} \text{ for } \text{v}_i;
                        \end{itemize}
                \end{itemize}
            \item \text{else} 
                \begin{itemize}
                    \item add \text{v}_i(\text{m}) \text{ to } \text{I};
                \end{itemize}
        \end{itemize}
\end{itemize}

\text{sumInfluence} = \text{sum of all membership values in } \text{I};

\text{Emit}(0,0 \mid \text{I} \text{ } \text{null}; \text{null})
\end{algorithm}

\section*{Algorithm 2: Reducer Part}
\begin{algorithm}
\textbf{Method Reduce} (Key k, [\text{I}_{temp,0} \mid \text{sumInfluence}_0, \ldots, \text{I}_{temp,J} \mid \text{sumInfluence}_J])
\begin{itemize}
    \item \text{max} = 0, \text{id}_{\text{max}} = \text{null}, \text{l}_{\text{max}} = \text{null};
    \item \text{for each } \text{id}_i \text{ in } \text{I}_{temp,i} \text{ do}
        \begin{itemize}
            \item \text{if} \text{sumInfluence}_i > \text{max} \text{ then}
                \begin{itemize}
                    \item \text{max} = \text{sumInfluence}_i;
                    \item \text{id}_{\text{max}} = \text{id}_i;
                    \item \text{l}_{\text{max}} = \text{I}_{temp,i};
                \end{itemize}
            \end{itemize}
    \item \text{Add} \text{id}_{\text{max}} \text{ to the set } S;
    \item \text{l} = \text{l}_{\text{max}};
    \item \text{if} \text{max} < \text{N} \times \text{x}\% \text{ then}
        \begin{itemize}
            \item Increment a predefined job counter called \text{numberOfIterations};
        \end{itemize}
\end{itemize}
\end{algorithm}

If the job counter \text{numberOfIterations} is greater than 0, then the above MapReduce algorithm will run again to add the next most influential vertex to \text{S}.

\section*{6. Conclusion}
In this paper, we propose a fuzzy propagation model to simplify the description of how a vertex influences others in a large-scale weighted social network. A MapReduce algorithm was then designed to locate individual zone for each vertex of the network. The concept of individual zone approximates the influence propagated from a vertex by using an \text{\alpha}-cut fuzzy set. Then, a MapReduce-based greedy algorithm was designed to identify MIV from all individual zones.

\section*{Acknowledgement}
This research is partly supported by the Kennesaw State University College of Science and Mathematics Interdisciplinary Research Opportunities (IDROP) program.

\section*{References}