Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2017

Hide and Seek in a Social Network

Olle Abrahamsson

Master of Science Thesis in Electrical Engineering

Hide and Seek in a Social Network

Olle Abrahamsson

LiTH-ISY-EX--17/5038–SE url

Supervisor: **Erik G Larsson**
ISY, Linköpings universitet

Examiner: **Danyo Danyev**
ISY, Linköpings universitet

Division of Communication
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Copyright © 2017 Olle Abrahamsson

Abstract

In this thesis a known heuristic for decreasing a node's centrality scores while maintaining influence, called ROAM, is compared to a modified version specifically designed to decrease eigenvector centrality. The performances of these heuristics are also tested against the Shapley values of a cooperative game played over the considered network, where the game is such that influential nodes receive higher Shapley values. The modified heuristic performed at least as good as the original ROAM, and in some instances even better (especially when the terrorist network behind the World Trade Center attacks was considered). Both heuristics increased the influence score for a given targeted node when applied consecutively on the WTC network, and consequently the Shapley values increased as well. Therefore the Shapley value of the game considered in this thesis seems to be well suited for discovering individuals that are assumed to actively trying to evade social network analysis.

Acknowledgements

Firstly I would like to thank my supervisor professor Erik G. Larsson for his never ending enthusiasm for this subject and all the support he has given me, and for introducing me to the wonderful subject of network theory. Likewise I would like to thank my examiner, docent Danyo Danyev, for taking the time and interest in this thesis.

Secondly, many thanks to my opponent Andreas Christensen for feedback and valuable criticism.

Finally I am very grateful to my friend Christoffer Holm whose knowledge in programming led to many valuable suggestions during the implementation phase of the project.

Contents

List of Figures

List of Tables

Nomenclature

Most of the recurring letters and symbols are described here.

Letters

G	Graph, network
v, $v^{\dagger}$, v_i, for $i \in \mathbb{N}$	Vertices
$v_i v_j$, for $i, j \in \mathbb{N}$	Edge between v_i and v_j

Symbols

$C_D(v)$	Degree centrality
$C_C(v)$	Closeness centrality
$C_B(v)$	Betweenness centrality
$C_E(v)$	Eigenvector centrality
$C_{D_m}(v_i)$	mth order degree mass
$V(G)$	The set of vertices of a graph G
$E(G)$	The set of edges of a graph G
$N(v)$	Neighbourhood of the vertex v

Other conventions

□ End of proof

1 Introduction

We live in a global world that is becoming increasingly interconnected, which has spawned significant interest in social network analysis (SNA). This has lead to a rapid development of tools to analyse our behaviour online, and among many other utilities these tools can be used in the prevention and investigation of terrorists and other benevolent actors. Consequently, one must therefore assume that countermeasures are taken by the adversaries to evade detection from such analyses. In this thesis we study one such technique called ROAM, first developed by Waniek et al in [14], where it was shown how an influential individual or community can manage their connections in order to lower some of their centrality scores while maintaining a high degree of influence in the network.

1.1 Earlier work

A number of various countermeasures have been proposed to tackle the issue of online privacy, e.g. algorithmic solutions [3] or market mechanisms which enables social media users to monetize on their private information [5]. However, little attention has been given to the study of *evading* SNA, and instead most research has been focused on developing progressively more advanced analysis tools. In [9] a new paradigm for SNA is suggested, whereby the strategic behaviour of network actors is explicitly modelled, and in [14] two heuristics (of which one, ROAM, is already mentioned) are proposed for the intent of concealing individuals and communities, respectively.

1.2 Method

The contribution of this thesis is a modified version of the proposed ROAM algorithm, called ROAMeig, which takes into account information about the entire network, not just the individual's immediate neighbours. This is accomplished by replacing the degree centrality measure in ROAM with the second degree mass, a novel concept statistically correlated to the leading eigenvector of the adjacency matrix of the network. This first task is what the firs part of the title, *hide*, is referring to. The other part, *seek*, alludes to developing methods to counteract the evasive attempt. Therefore we also test how these methods for cloaking an important node stand up when we apply centrality analysis based on cooperative game theory. This is done by considering a cooperative game in which coalitions are highly rewarded if they maximise the social influence (defined as the number of agents in the coalition plus the number of agents outside of the coalition reachable by one hop.) The centrality of each node is then the node's Shapley value for the considered game. If the network under analysis is disconnected, the Myerson value is used instead.

1.3 Demarcation

We will restrict our study to undirected and unweighted networks without self-loops. This is because we are only interested in who knows who, and we make no assumptions on the strength of these connections, so weights are not needed. In this context self-loops makes little sense since the edges represent friendship or acquaintance. Both when considering the influence models in Chapter 2 and the cooperative games in Chapter 4, we assume that the agents represented by the nodes act rationally. From a game theoretical viewpoint this means that players strive to maximise their long term payoff.

1.4 Thesis layout

The thesis is structured as follows: In Chapter 2 we introduce the reader to some basic definitions and facts about graphs and network theory, especially centrality and influence metrics. In Chapter 3 the original and modified algorithms are described and discussed. Chapter 4 introduces a recently developed class of centrality measures based on cooperative game theory. In Chapter 5 some experimental results are reported where comparisons are made between the original and modified algorithms. Their abilities to cloak the targeted node is also tested with the respect to discovering this node via the game theoretic centrality analysis introduced in Chapter 4. The results are discussed and commented on in Chapter 6. Finally, in Chapter 7 some ideas for further research are suggested. There are also two appendices: In Appendix A some supplementary theory is provided, and in Appendix B the source code for the implementations in Python are listed.

2

Preliminaries

This chapter will define notions used in the thesis. For a more thorough introduction to graph theory, see for instance [1].

2.1 Graph theory

The following definitions are taken from [2]. Our main objects of study are graphs, which are often called a network in applied contexts.

Definition 2.1. A **graph** G is a pair $(V(G), E(G))$ consisting of a set $V(G)$ of **vertices** (or *nodes*) and a set $E(G)$ of edges, where each edge connects two distinct vertices and no two vertices are connected by more than one edge. Two vertices are **adjacent** if they are connected by an edge.

Remark 2.2. We will often say that two adjacent vertices are **neighbours**. We will also use the terms *graph* and *network* interchangeably when suited.

Given a graph, we can take subsets of its vertex and edge sets and form a new graph, called a subgraph. This will be helpful in Chapter 4 when we will play games on such subgraphs (called coalitions in the language of cooperative game theory).

Definition 2.3. A graph G' is a **subgraph** of the graph G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$.

One of the most important features of a node in network theory is its degree, which tells us how many friends the person represented by the node has in the network. These friends constitute the node's neighbourhood.

Definition 2.4. The **degree** of a vertex $v \in V(G)$, denoted $\boldsymbol{d(v)}$, is the number of edges that are incident with v.

Definition 2.5. The **neighbourhood** of a vertex v, denoted $\boldsymbol{N(v)}$, is the set of vertices adjacent to v.

Two other concepts that are of great importance for us are *path* and *distance* which give us information about how many hops are needed to reach a certain node from a given starting node.

Definition 2.6. A **path**, denoted by P, is a non-empty graph or subgraph of the form $P = (V, E)$, with $V = \{v_0, \ldots, v_n\}$ and $E = \{v_0v_1, \ldots, v_{n-1}v_n\}$ where all v_i are distinct. We say that P joins the vertices v_0 and v_n, or that P is a $\boldsymbol{v_0 - v_n}$ **– path**. The number of edges in a path is called the **length** of a path.

Definition 2.7. The **distance** between two vertices u and v, denoted by $\boldsymbol{d(u, v)}$, is the length of the shortest path joining u and v. If u and v are not connected by any path, then $d(u, v)$ is infinite.

Lastly we shall define the adjacency matrix of a network. This is an algebraic representation of a graph which enables us to utilise all tools from linear algebra and matrix analysis and apply them to the study of the network in question.

Definition 2.8. The **adjacency matrix** A of a graph G is a matrix of the form (a_{uv}) where

$$a_{uv} = \begin{cases} 1, \text{ if } u \text{ and } v \text{ are neighbours} \\ 0, \text{ otherwise .} \end{cases}$$

Remark 2.9. In this thesis we will only consider undirected networks, that is, $v_iv_j \in E(G)$ is equivalent to $v_jv_i \in E(G)$, so we let $v_iv_j = v_jv_i$. In the following, we will therefore assume that the adjacency matrix A is symmetric, so that $A = A^T$.

These concepts will hopefully become clear (if they are not already) if we put them in context with an example, so let us do that.

Example 2.10

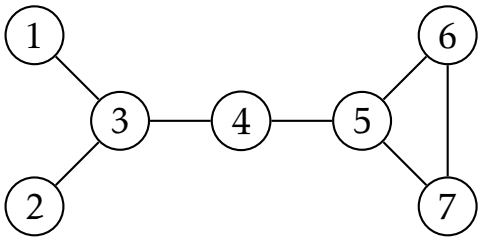


Figure 2.1: *An example of a graph, G, with 7 nodes and 7 edges.*

The graph G in Figure 2.1 has the following properties. It is a graph with $|V(G)| = 7$ nodes and $|E(G)| = 7$ edges. In Figure 2.2 we see a subgraph G' with vertex set

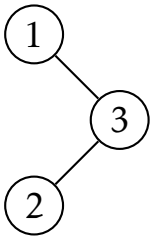


Figure 2.2: *A subgraph, G', of the graph G.*

$\{v_1, v_2, v_3\} = V(G') \subset V(G)$ and edge set $\{v_1v_3, v_2v_3\} = E(G') \subset E(G)$. The degree of node v_5 is $d(v_5) = 3$, its neighbourhood is $N(v_5) = \{v_4, v_6, v_7\}$.

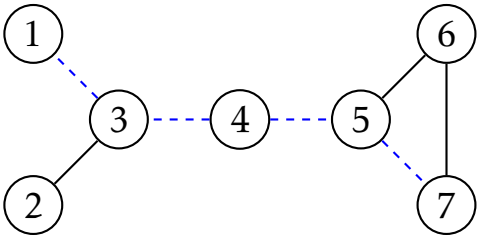


Figure 2.3: *The graph G, with a path (blue dashed edges) from v_1 to v_7.*

In Figure 2.3 we also see a path $v_1v_3v_4v_45_7$, with dashed blue edges, from node v_1 to node v_7, and since this happens to be the shortest path between these nodes, they have distance $d(v_1, v_7) = 4$. (Note that one could for instance have taken the alternative path $v_1v_3v_4v_5v_6v_7$, but that path has one more edge than the one illustrated, so it is not the shortest path.) Finally, the adjacency matrix for the graph G is

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

2.2 Centrality metrics

Indicators of centrality identify the most important vertex in some sense (depending on the measure chosen). In our context this simply means the most important individual with respect to the surrounding social network. It is worth pointing out that there are many other definitions of centralities than those listed here. However, the following are all we need for the task at hand. Let $G = (V(G), E(G))$ be a graph.

Definition 2.11. The **degree centrality** of a vertex v is

$$C_D(v) = d(v).$$

Definition 2.12. The **closeness centrality** of a vertex v is

$$C_C(v) = \frac{|V(G)|}{\sum_u d(u,v)},$$

where $d(u,v)$ is the distance between vertices u and v.

Remark 2.13. Note that since we are only considering (strongly) connected networks, the distance between any two nodes in the network is always finite, and so the denominator in the above definition is finite.

Definition 2.14. The **betweenness centrality** of a vertex v is

$$C_B(v) = \sum_{s \neq t \neq v \in V(G)} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the total number of distinct shortest paths from vertex s to vertex t, and $\sigma_{st}(v)$ is the number of those paths that pass through v.

These three centralities have fairly straightforward interpretations. *Degree centrality* is simply the number of edges that a vertex is connected to, *closeness* is the reciprocal of the average shortest distance from a vertex to all other nodes, and *betweenness* is basically the proportion of all possible shortest paths which also pass through the given vertex.

Example 2.15

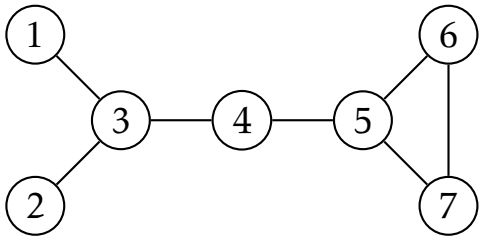


Figure 2.4: *An example of a network to illustrate the various centrality measures.*

Consider the network in Figure 2.4. The degree centralities are

$$C_D(3) = C_D(5) = 3$$
$$C_D(4) = C_D(6) = C_D(7) = 2$$
$$C_D(1) = C_D(2) = 1.$$

Thus it might be reasonable to claim that nodes 3 and 5 are the most *popular* in the network. On the other hand, if we calculate the closeness centralities, we find that

$$C_C(4) = 7/10$$
$$C_C(3) = C_C(5) = 7/11$$
$$C_C(6) = C_C(7) = 7/15$$
$$C_C(1) = C_C(2) = 7/16,$$

so node 4 could be considered the one which can *most efficiently obtain information* from every other node. Finally, if we instead are interested in betweenness centrality, we obtain that

$$C_B(3) = 16/3$$
$$C_B(4) = C_B(5) = 13/3$$
$$C_B(1) = C_B(2) = C_B(6) = C_B(7) = 0,$$

from which we can say that node 3 is the one that can most frequently control information flow in the network.

As we can see, the nodes are ranked differently depending on what metric is used. The situation for the example network is summarised in Table 2.1.

Rank	$C_D(v)$	$C_C(v)$	$C_B(v)$
1	3,5	4	3
2	4,6,7	3,5	4,5
3	1,2	6,7	1,2,6,7
4		1,2	

Table 2.1: *Node ranking with respect to degree, closeness and betweenness centralities for the example network.*

Another very common centrality measure is the *eigenvector centrality*, defined below, which is based on the eigenvectors of the adjacency matrix. It assigns relative scores to all of the nodes such that a vertex gets a high score if it is connected to other highly-scored vertices. Note how this recursion captures features of the global topology of the network, whereas the other three centrality measures only is influenced by the local topology.

Definition 2.16. The **eigenvector centrality** of a vertex v_i is recursively defined as

$$C_E(i) = \frac{1}{\lambda} \sum_k a_{k,i} C_E(k),$$

where λ is the largest eigenvalue in magnitude and $a_{i,j}$ are the elements of the adjacency matrix A. In matrix form this can be written as

$$A\overline{x} = \lambda \overline{x}.$$

Then $C_E(i) = \overline{x}(i)$, the ith element of the eigenvector $\overline{x}$.

Remark 2.17. By choosing λ to be the largest eigenvalue in magnitude, it follows from Perron-Frobenius theorem that if the matrix A is irreducible, or equivalently if the graph it represents is connected[1], then the eigenvector solution $\overline{x}$ is both unique and positive.

We also define a new class of centrality measures, the *degree mass*, which is a variant of degree centrality. This idea was introduced by Li et al in [7, p. 3].

Definition 2.18. The ***m*th order degree mass** of a vertex v_i is

$$C_{D_m}(v_i) = \sum_{k=1}^{m+1} (A^k u)_i = \sum_{j=1}^{N} \left(\sum_{k=0}^{m} A^k \right)_{ij} d(v_j),$$

where A is the adjacency matrix, $u = (1, 1, \ldots, 1)^T$ and $N = |V(G)|$.

As noted in [7, p. 7], the second order degree mass, that is when $m = 2$, correlates strongly with the components of the leading eigenvector of the adjacency matrix, and could therefore be a suitable approximation of eigenvector centrality since degree centrality has a linear time complexity, $\mathcal{O}(|V(G)|)$. The correlation between

[1] For directed graphs the condition is *strongly* connected. In this thesis directed graphs are not treated, but the interested reader is referred to e.g. [1].

$C_D(v)$ and $C_E(v)$ is further supported both numerically and analytically, via the central limit theorem, in [11]. Given a vertex v_i, we have for $m = 2$ that

$$\begin{aligned} C_{D_2}(v_i) &= \sum_{j=1}^{N} \left(\sum_{k=0}^{2} A^k \right)_{ij} d(v_j) \\ &= \sum_{j=1}^{N} \left(I + A + A^2 \right)_{ij} d(v_j), \end{aligned} \tag{2.1}$$

which can be interpreted as a weighted sum of all degrees.

2.3 Influence metrics

A concept closely related to that of centrality is the *influence* of a vertex. In our context of social networks, this corresponds to how much influence an individual has on the other individuals in the network. Since our aim is to lower centrality but leave influence mainly unaffected, it is of great importance to have a clear definition of how to measure this property.

When a vertex is sufficiently influenced by its neighbour(s), we say that it becomes *active*, at which point it starts to influence any neighbour that has not yet become active. In order to initiate the process a set of vertices are activated from the start. This set is called the *seed set*.

Definition 2.19 (Active set). The subset of vertices that are active at time t is called the *active set* and is denoted by

$$I(t) \subseteq V(G), \quad t = 0, 1, \dots .$$

The active set $I(0)$ is called the *seed set*.

Active vertices influence inactive vertices differently depending on which influence metric is considered. We will use two metrics, *independent cascade* and *linear threshold*.

Definition 2.20 (Independent cascade). To every pair of vertices we assign an activation probability

$$p : V(G) \times V(G) \to [0, 1].$$

Then at every time $t \geq 1$, every vertex $v \in V(G)$ that became active at time $t - 1$ activates every inactive neighbour, $w \in N(v) \setminus I(t-1)$, with probability $p(v, w)$. The process ends when $I(t) = I(t-1)$ (i.e. when there are no new active vertices).

Definition 2.21 (Linear threshold). To every vertex $v \in V(G)$ we assign a threshold value t_v which is sampled from the set $\{0, \dots, |N(v)|\}$ according to some probability distribution. Then, at every time $t \geq 1$, every inactive vertex v becomes active if $|I(t-1) \cap N(v)| \geq t_v$. The process ends when $I(t) = I(t-1)$.

To better understand the linear threshold model, let us look at an example.

Example 2.22

Consider again the network in Figure 2.5, now with added threshold values on each node, and the seed node marked with red. In the first iteration, shown in Figure 2.6 we see that the target node's neighbours have thresholds $t_3 = 0$ and $t_5 = 1$, so both get activated. In the next iteration, shown in Figure 2.7, all nodes except node 7 gets activated. This is because node 7 needs at least two active neighbours to become activated itself, but at this step only one neighbour, node 5, is currently active. In the third and final iteration, however, node 6 is now newly activated, so node 7 has two active neighbours and thus becomes activated itself. This is illustrated in Figure 2.8. The process stops since the active set of nodes cannot grow any further, i.e. $I(4) = I(3)$. Note however that it is only a coincidence that every node in the network became activated – this is typically not the case, as we shall see later on in Chapter 5.

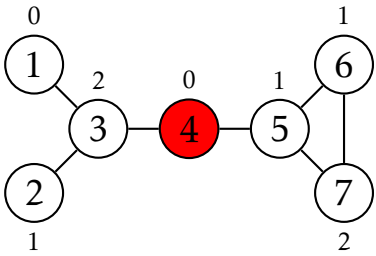


Figure 2.5: *An example of a network to illustrate the linear threshold influence model.*

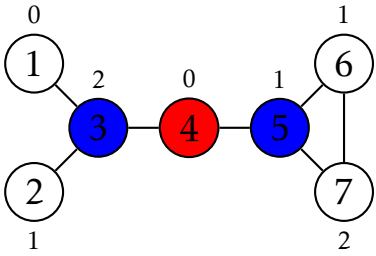


Figure 2.6: *The network after one iteration of linear threshold.*

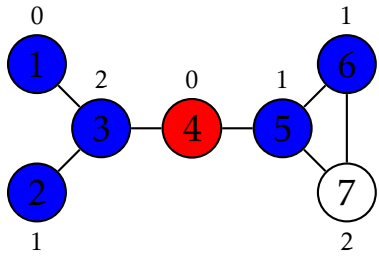


Figure 2.7: *The network after two iterations of linear threshold.*

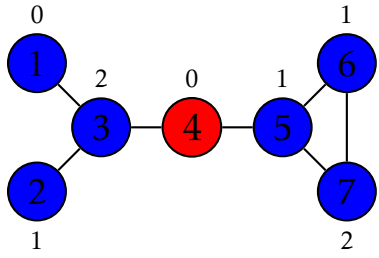


Figure 2.8: *The network after three iterations of linear threshold. In this case all nodes became activated.*

3

The ROAM heuristic

In this chapter we will describe a heuristic developed by Waniek et al [14, pp. 3-5] called ROAM (Remove One, Add Many), which is used for concealing a person while maintaining their influence over the community to which the person belongs. Given a target vertex (typically a leader of the community), the idea is to disconnect the target from one of its neighbours and then connect this neighbour to some of the target's other friends, where the number of edge modifications allowed is restricted. A modified version of ROAM adapted for the eigenvector centrality measure will also be suggested.

3.1 Original ROAM

Let G be a network with $v^\dagger = \max_{v \in G} C_D(v)$, and let $b \in \{1, 2, \dots\}$ be the number of edges that we allow to be removed or added. We will henceforth refer to b as the *budget* of ROAM.

There are basically two critical steps in ROAM. The first is when the edge between the target vertex $v^\dagger$ and the neighbour v_0 is removed. Note that this removal can only decrease the degree of $v^\dagger$. It also decreases the closeness and betweenness of $v^\dagger$ since it removes any shortest path between $v^\dagger$ and any other vertex which runs through this edge. On the downside, $v^\dagger$ no longer has any direct influence over v_0. This is remedied in the second step, where v_0 is connected to the $b - 1$ other friends of $v^\dagger$ with the lowest degrees. These new, indirect, paths between $v^\dagger$ and v_0 compensates for the lost direct path. This step does not affect the degree of $v^\dagger$, but it does of course increase the degrees for some of its neighbours, which further disguises the importance of $v^\dagger$. The closeness and betweenness of $v^\dagger$ cannot increase either. To see this, suppose that $v_i \in N(v^\dagger)$ with $v_i \neq v_0$.

Algorithm 1 The ROAM heuristic

Input: $G, b, v^\dagger$
Output: A network G'
$r \leftarrow \left|N(v^\dagger)\right| - 1$
$S \leftarrow \{v_0, v_1, \ldots, v_r\}$ such that $C_D(v_0) \geq C_D(v_1) \geq \cdots \geq C_D(v_r)$, for all $v_i \in N(v^\dagger), \quad i = 0, \ldots, r$
if $r \geq b - 1$ **then**
    $E(G') \leftarrow \left(E(G) \setminus \{v^\dagger v_0\}\right) \cup \{v_0 v_{r-(b-1)}, v_0 v_{r-(b-2)}, \ldots, v_0 v_r\}$
else
    if $r < b - 1$ **then**
        $E(G') \leftarrow \left(E(G) \setminus \{v^\dagger v_0\}\right) \cup \{v_0 v_1, v_0 v_2, \ldots, v_0 v_r\}$
return $G' = (V(G), E(G'))$

Then any path containing the edges $v^\dagger v_i$ and $v_i v_0$ is longer than the direct path $v^\dagger v_0$ which was removed in step one, and this does not increase the percentage of shortest paths going through $v^\dagger$.

Thus the overall effect of the heuristic is that the degree, closeness and betweenness centralities of $v^\dagger$ are reduced while most of its influence over $N(v^\dagger)$ is preserved.

Example 3.1

Consider the network in Figure 3.1(a) and suppose that our budget is $b = 4$. In this case we assume that $v^\dagger$ is the target because it has the highest degree centrality, which suggests that it might be a leader in this social community. First we localise its neighbours with the highest degree centrality (note that there may be several vertices with the same maximum degree). They are v_0, v_1, v_2 and v_4, all with degree 3. We choose one of them, say v_0, and remove the edge between $v^\dagger$ and v_0.

Now we shall connect v_0 to the $b - 1 = 3$ nodes in $N(v^\dagger)$ with lowest degree centralities. We have $N(v^\dagger) \setminus v_0 = \{v_1, v_2, v_3, v_4, v_5\}$ with $C_D(v_5) = 1$, $C_D(v_3) = 2$ and $C_D(v_i) = 3$ for $i = 1, 2, 4$. When deciding which of these nodes to connect to v_0, we naturally include v_5 and v_3 since they have the lowest degrees of all the considered neighbours, and we pick one of $\{v_1, v_2, v_4\}$ arbitrarily, say v_4.

Repeating the entire procedure results in the network shown in Figure 3.1(c).

3.2 Modified ROAM

In this modified version, we would like to somehow replace the degree centrality $C_D(v)$ with the eigenvector centrality $C_E(v)$. Here is where the degree mass (see Definition 2.18) comes in handy. Recall that the second degree mass was

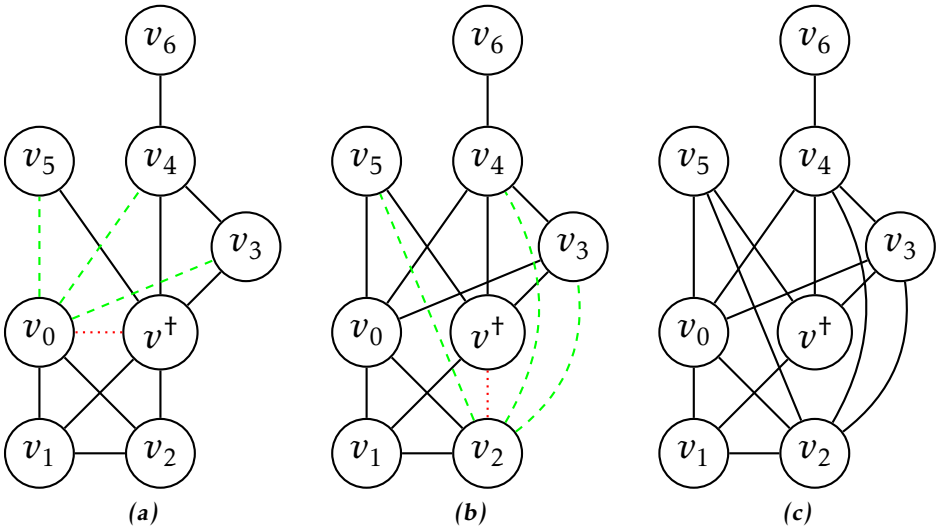


Figure 3.1: *The original ROAM heuristic with budget $b = 4$ applied to a small network where the dotted red edge is removed, and the dashed green edges are added. (a) The original network. (b) The network after one round of ROAM. (c) The network after two rounds of ROAM.*

highly correlated with the eigenvector centrality as we discussed earlier. Consequently we expect that if $C_D(v_i)$ is replaced with $C_{D_2}(v_i)$, then the ROAM algorithm would lower $C_E(v_i)$ as well. Thus we propose the following modification:

In view of Equation (2.1), the only practical difference from the original ROAM in terms of calculations is the extra cost of calculating A^2, which has a time complexity of $\mathcal{O}(|V(G)|^2)$. While this is relatively expensive operation, it is only performed once. Thus it should not affect the computational time severely. It might put greater demands on the amount of data that has to be kept in memory.

Algorithm 2 The modified ROAM heuristic

1: Input: $G, b, v^{\dagger}$
2: Output: A network G'
3: $r := \left|N(v^{\dagger})\right| - 1$
4: $S \leftarrow \{v_0, v_1, \ldots, v_r\}$ such that $C_{D_2}(v_0) \geq C_{D_2}(v_1) \geq \cdots \geq C_{D_2}(v_r)$, for all $v_i \in N(v^{\dagger}), \quad i = 0, \ldots, r$
5: **if** $r \geq b - 1$ **then**
6: $\quad E(G') \leftarrow \left(E(G) \setminus \{v^{\dagger}v_0\}\right) \cup \{v_0 v_{r-(b-1)}, v_0 v_{r-(b-2)}, \ldots, v_0 v_r\}$
7: **else**
8: $\quad$ **if** $r < b - 1$ **then**
9: $\quad\quad E(G') \leftarrow \left(E(G) \setminus \{v^{\dagger}v_0\}\right) \cup \{v_0 v_1, v_0 v_2, \ldots, v_0 v_r\}$
10: **return** $G' = (V(G), E(G'))$

4

A game theoretic approach

So far we have seen how an individual can evade conventional social network analysis with respect to the most common centrality metrics. One must therefore ask if it is still possible to detect these hidden individuals by other means. In this chapter we will discuss one such approach based on cooperative game theory, the study of games in which the participants form coalitions with each other, and compete against other coalitions in the same network. The cooperation is enforced on the players in forms of payoffs, so that certain alliances are more highly rewarded than others. We will first introduce a rather general concept, and then consider specialised methods applicable to social networks.

4.1 Cooperative games

Let G be a network. In this context the set of vertices $V(G)$ are called *players*, denoted by V. A subset $S \subseteq V$ of the set of players is called a *coaltition*.

Definition 4.1 (Characteristic function). Let V be a set of players. The *characteristic function*, denoted, $v_G(S)$ is a function

$$v : 2^V \to \mathbb{R}$$

which assigns a *payoff* to every coalition $S \subseteq V$. The empty set, $\emptyset$, has the payoff value $v(\emptyset) = 0$.

A cooperative game is a pair (V, v) of a set of players and a characteristic function, and a solution to such a game is a strategy to divide $v(V)$ – the payoff from cooperation – among the players. One such strategy is the Shapley value, introduced in 1953 by Lloyd Shapley in [12].

Definition 4.2. The *Shapley value* for a player $i \in V$ is defined by

$$SV_i(v) = \sum_{S \subseteq V \setminus \{i\}} \xi_S(v(S \cup \{i\}) - v(S)),$$

for some characteristic function v, where

$$\xi_S = \frac{|S|!(|V| - |S| - 1)!}{|V|!}.$$

This can be interpreted as a weighted average marginal contribution of a player to every coalition S this player could belong to. That is,

$$SV_i(v) = \sum_{\text{coalitions excluding } i} \frac{\text{marginal contribution of } i \text{ to coalition}}{\text{number of coalitions of this size, excluding } i},$$

divided by the number of players in the network. The Shapley value is important since it is the unique strategy which exhibits the following four desirable properties:

(i) The value of a coalition, $v(V)$, equals the sum of the payoffs of the players in that coalition, so the strategy is efficient

(ii) Symmetric players obtain symmetric payoffs[1]

(iii) Agents not contributing to the game obtain no payoff, and

(iv) The division scheme is additive[2]

We want to restrict these games to networks, but we still want to allow coalitions to be formed by sets of disjoint, connected components of the network. Such coalitions are said to be *disconnected*. In order to accomplish this, Myerson [10] considered a characteristic function defined over both connected and disconnected coalitions, which makes use of the notion of induced subgraphs.

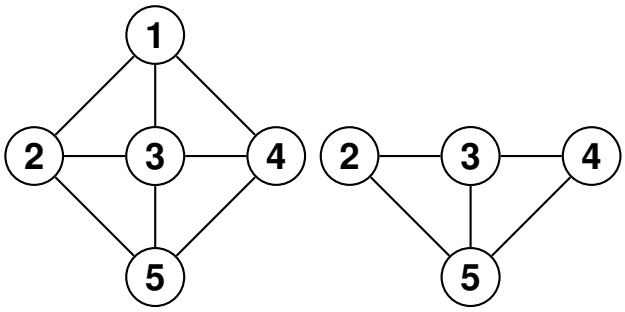
Definition 4.3. Let $G(V, E)$ be a graph, and consider a subset of vertices $V' \subseteq V$. The *induced subgraph* $G[V']$ is the graph whose vertex set is V' and whose edge set, E', consists of all edges in E such that both endpoints are vertices in V'.

Example 4.4

In Figure 4.1(b) we see the subgraph of (a) induced by the vertex subset $\{2, 3, 4, 5\}$.

[1] This simply means that players that are strategically equivalent in the game receive the same payoff – they are symmetric.

[2] In other words, a transfer of an amount x from one player to another *decreases* the first player's utility by x units, and *increases* the second player's utility by x units. Consequently the players' individual payoffs add up to the payoff of the coalition they are members in – the division scheme is additive.



(a) *A graph G.* ***(b)*** *The induced subgraph $G[\{2,3,4,5\}]$.*

Figure 4.1: *A graph G and an induced subgraph $G[\{2,3,4,5\}]$.*

Now we are ready to talk about the characteristic function considered by Myerson, which is defined by

$$v_G^{\mathcal{M}}(S) = \begin{cases} v(S) & \text{if } S \in \mathcal{C}(G) \\ \sum_{K_i \in K(S)} v(K_i) & \text{otherwise,} \end{cases}$$

where $\mathcal{M}$ stands for Myerson[3], $\mathcal{C}(G)$ is the set of all connected induced subgraphs of G, and $K(S) = \{K_1, K_2, \ldots, K_m\}$ is the set of connected components of coalition S if it is disconnected. This means that the payoff of a disconnected coalition is the sum of the payoffs of its constituent components. Note that $v_G^{\mathcal{M}}$ is defined over all $2^{|V|}$ coalitions, so the Shapley value can be applied. This is not the case with the simpler characteristic function $v_G : \mathcal{C}(G) \to \mathbb{R}$, defined by $v_G(S) = v(S)$ for all coalitions $S \in \mathcal{C}(G)$, which $v_G^{\mathcal{M}}$ is a generalisation of. This is of course due to the fact that we only consider connected coalitions in the simpler function. However, Myerson famously proposed a solution for this obstacle in [10], namely the following. First, we define the *Myerson value*.

Definition 4.5. The *Myerson value* for a player $i \in V$ is defined by

$$MV_i(v_G) = SV_i(v_G^{\mathcal{M}}),$$

where $SV_i(\cdot)$ is the Shapley value.

Now, let each player v_i in coalition V be assigned the payoff $MV_i(v_G)$, i.e. the Myerson value. Then we obtain the unique payoff division scheme that is characterised by two axioms: efficiency by components (the payoff vectors are efficient[4] in each maximal connected coalition for each network) and fairness (the loss of one bilateral communication implies the same loss of payment for the players involved in this edge).

[3] This is just a notation in order to differentiate it from other characteristic functions; it has no special technical meaning.

[4] This is property (i) above.

4.2 One game to consider

A reasonable game for our purpose – finding hidden individuals with strong influence – must be one that captures the social influence. Let's consider the following game, denoted by g_1 and suggested by Suri and Narahari [13], in which the value of a group of players is the number of players within and adjacent to the group. It is intuitively clear that the Shapley value of players in this game defines a centrality measure that is qualitatively better at capturing the influence property than the ones we studied in earlier chapters. Let us illustrate this argument with a small example network.

Example 4.6

Consider the network in Figure 4.2. While nodes v_1 and v_5 both have degree $C_D(v_1) = C_D(v_5) = 3$, node v_5 will receive a higher Myerson (or Shapley) value in the considered game since it is the only player than can influence node v_6. Therefore, with respect to influence, it seems reasonable to rank v_5 higher than v_1. This interesting feature – which is obvious in this small example – would not have been captured in the traditional centrality rankings.

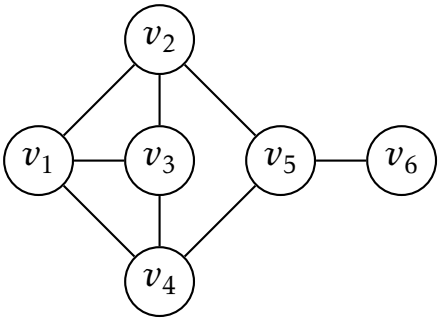


Figure 4.2: *An example of a network to illsutrate the game g_1.*

A mathematical deduction of an algorithm for calculating the Shapley values for this game is given in Appendix A.

5

Experiment and results

5.1 Data sets

The algorithms are tested on several different networks, all simple, undirected and without self-loops. The first two are real networks:

- Covert organizations: This category is represented by the terrorist network responsible for the attacks on the World Trade Center (WTC), September 11, 2001 [4].
- Social networks: We studied an anonymized fragment of Facebook, taken from SNAP – the Stanford Network Analysis Platform [6].

We also study randomly-generated networks:

- Two scale-free networks based on the Barabási-Albert (BA) model (for a technical explanation of this model, see Appendix A.2).

The following table contains basic data about the four networks.

Network	Nodes	Edges
Facebook	333	2519
WTC	60	124
Small scale-free	200	1945
Large scale-free	1000	9945

5.2 Experiments with ROAM and modified ROAM

From the results in Waniek et al [14], it is clear that a higher budget b gives better results in general. However, for $b > 4$ the improvement seems to be negligible. For the purpose of comparing the original ROAM with the modified variant (hereinafter called ROAMeig) we fix the budget to $b = 3$, and the heuristic with this budget will be denoted by ROAM(3) and ROAMeig(3), respectively. This value is chosen since it is large enough to show interesting features of the heuristics, and is thus suitable for comparisons. The figures 5.1–5.4 below show how a chosen target node (in these examples the target node is the one with highest degree centrality – picked arbitrarily if there are several candidates[1]) is affected with respect to betweenness, closeness and eigenvector centralities when ROAM(3) and ROAMeig(3) is run 8 consecutive times for each network. In Figures 5.5–5.6 the Shapley values for the respective target nodes are shown for the four networks (one network per subfigure) and how they are affected by applying ROAM(3) and ROAMeig(3).

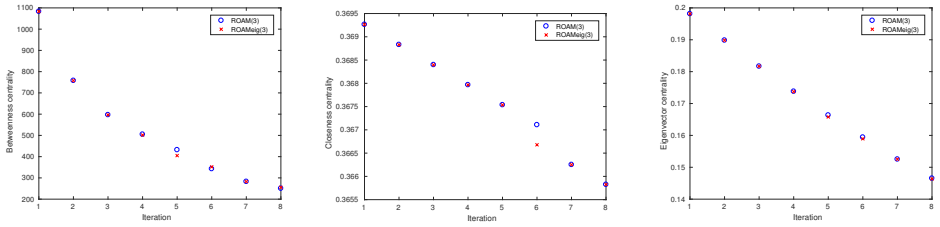


Figure 5.1: *ROAM(3) and ROAMeig(3) run on the Facebook network.*

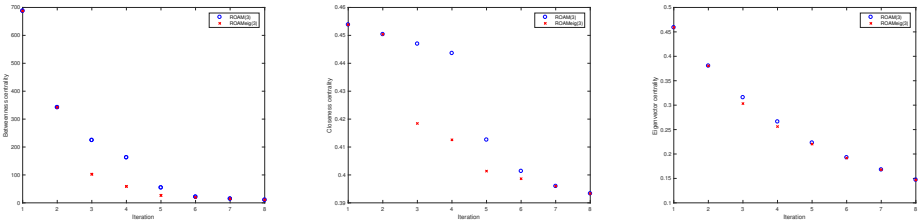


Figure 5.2: *ROAM(3) and ROAMeig(3) run on the WTC terrorist network. Mohamed Atta (node 26) was the target node.*

[1] The one exception is for the WTC terrorist network, where we know that the node 26, representing Mohamed Atta, was one of the ringleaders and is therefore a priori considered the target node in this network.

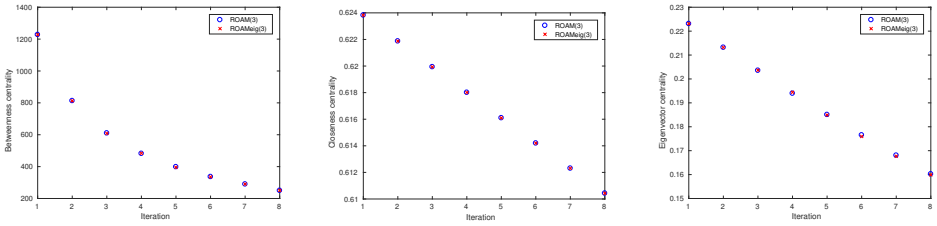


Figure 5.3: *ROAM(3) and ROAMeig(3) run on the smaller scale free network.*

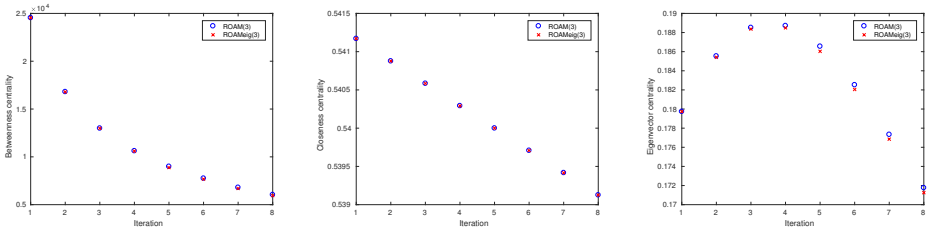


Figure 5.4: *ROAM(3) and ROAMeig(3) run on the larger scale free network.*

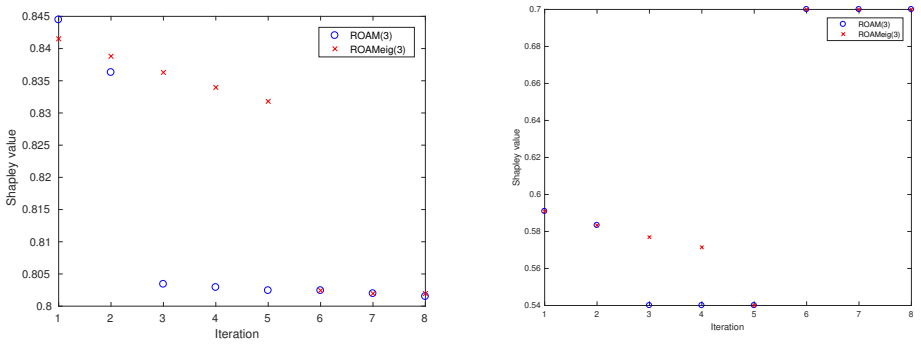


Figure 5.5: *ROAM(3) and ROAMeig(3) run on the Facebook network (left) and the WTC terrorist network (right).*

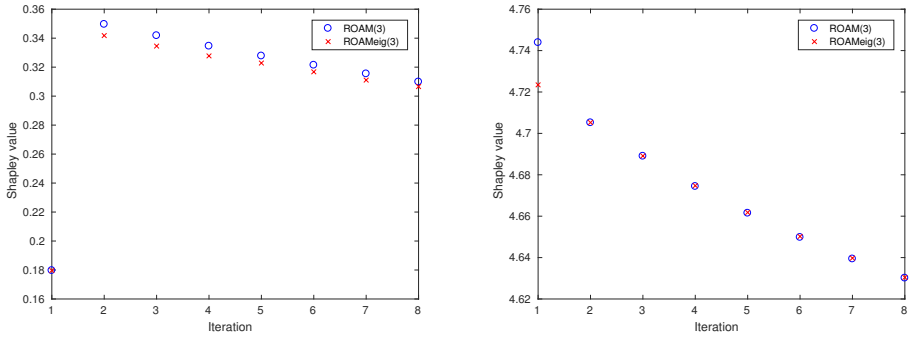


Figure 5.6: *ROAM(3) and ROAMeig(3) run on the smaller scale-free network (left) and the larger scale-free network (right).*

Tables 5.1–5.4 below show the number of nodes that became activated, first within the original networks, and also after three iterations of ROAM(3) and ROAMeig(3) according to the two influence models we are considering – linear threshold and independent cascade. Figures 5.7-5.10 illustrate the process for the WTC network with Mohamed Atta as the seed node. This particular network was chosen for the illustration since it is small enough to show a suitable level of detail. Each graph is the average of 10 runs of the respective model. The left graph in each pair shows the cascade before ROAM(3) or ROAMeig(3), and the right graph shows the cascade after the heuristics have been applied three consecutive times.

Influence model	Before	3×ROAM(3)	3× ROAMeig(3)
Linear threshold	42	42	57
Independent cascade	23	26	31

Table 5.1: *Number of activated nodes in the WTC terrorist network before and after applying three rounds of ROAM(3) and ROAMeig(3), respectively.*

Influence model	Before	3×ROAM(3)	3× ROAMeig(3)
Linear threshold	140	133	129
Independent cascade	257	272	262

Table 5.2: *Number of activated nodes in the Facebook network before and after applying three rounds of ROAM(3) and ROAMeig(3), respectively.*

Influence model	Before	3×ROAM(3)	3× ROAMeig(3)
Linear threshold	72	87	70
Independent cascade	200	199	199

Table 5.3: *Number of activated nodes in the smaller scale-free network before and after applying three rounds of ROAM(3) and ROAMeig(3), respectively.*

Influence model	Before	3×ROAM(3)	3× ROAMeig(3)
Linear threshold	409	390	355
Independent cascade	998	996	997

Table 5.4: *Number of activated nodes in the larger scale-free network before and after applying three rounds of ROAM(3) and ROAMeig(3), respectively.*

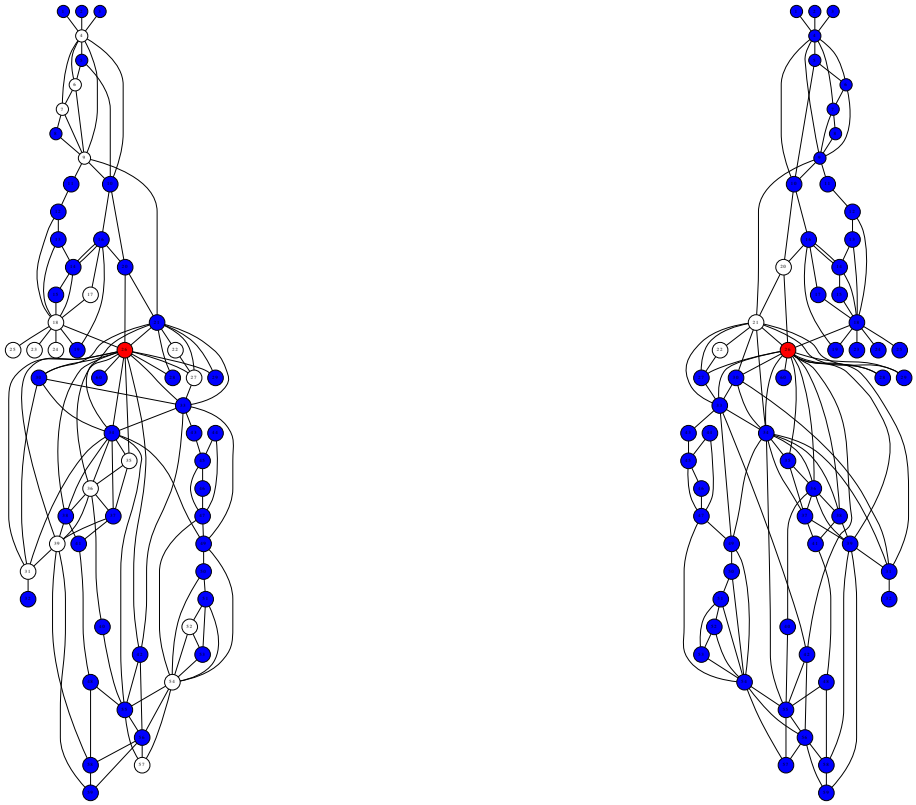


Figure 5.7: *Left: Linear threshold model of influence for the terrorist network responsible for the 9/11 WTC attacks with Mohamed Atta (node 26, in red) as seed node. Right: The same model and network, but after 3 rounds of ROAM.*

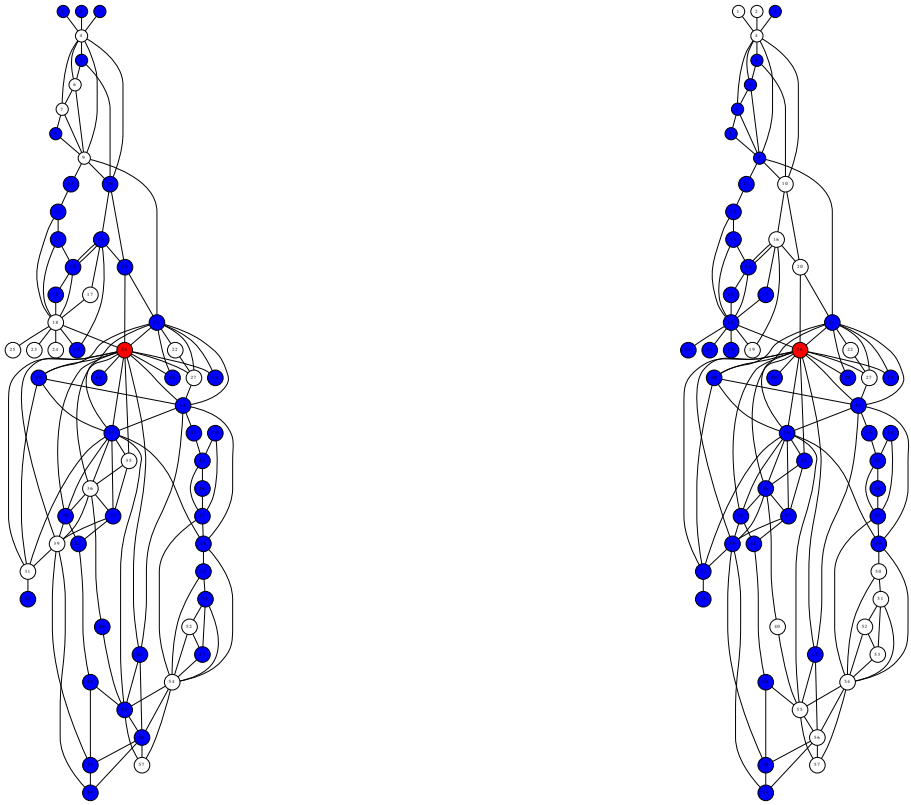


Figure 5.8: *Left: Linear threshold model of influence for the terrorist network responsible for the 9/11 WTC attacks with Mohamed Atta (node 26, in red) as seed node. Right: The same model and network, but after 3 rounds of ROAMeig(3).*

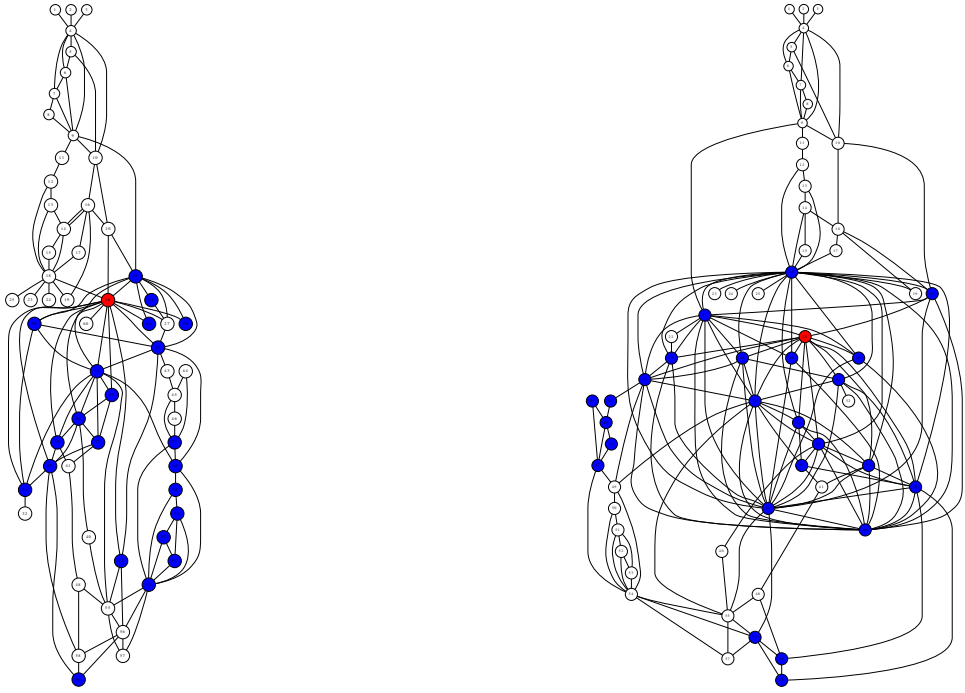


Figure 5.9: *Left: Independent cascade model of influence for the terrorist network responsible for the 9/11 WTC attacks with Mohamed Atta (node 26, in red) as seed node. Right: The same model and network, but after 3 rounds of ROAM(3).*

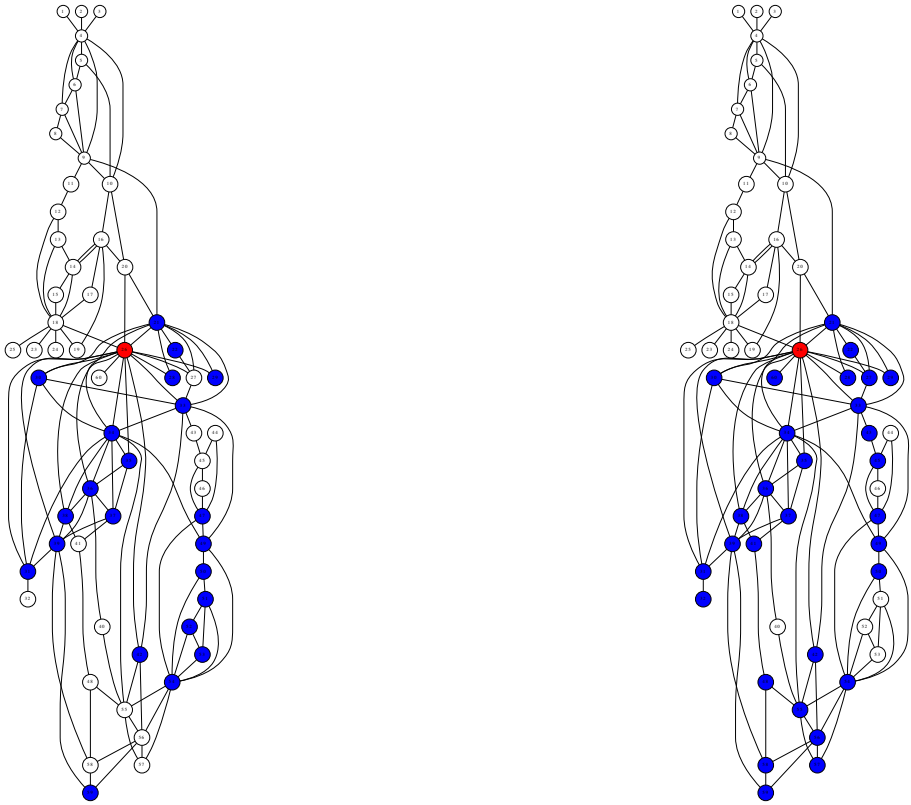


Figure 5.10: *Left: Independent cascade model of influence for the terrorist network responsible for the 9/11 WTC attacks with Mohamed Atta (node 26, in red) as seed node. Right: The same model and network, but after 3 rounds of ROAMeig(3).*

6

Discussion and conclusions

From the centrality value plots (Figures 5.1-5.4) it is clear that in general the ROAMeig algorithm performed just as well as the original ROAM, and for the WTC network it even seems to perform somewhat better than ROAM. There is no obvious reason as to why this should be the case (except for the eigenvector centrality) – it might just be a coincidence. Alternatively there is something about the topology for that exact network which plays a role here, but a more detailed analysis is needed before any conclusions can be drawn, and that lies outside of the scope of this thesis. The general similarity in performance of the two heuristics is arguably best explained by the fact that they are very similarly constructed, and the only difference is how to choose the target node's $b-1$ friends. The main improvement is with respect to eigenvector centrality, which is evident from the figures above. This should not be particularly surprising since ROAMeig was constructed with this very purpose in mind. The other decreased centrality values achieved by running ROAMeig can be explained by the same reasoning as in Section 3.1 since they work very similar in practice.

Regarding the influence, it is somewhat surprising that both models show an improved level of influence (in terms of number of successfully activated nodes). The increase of influence with respect to the linear threshold model is consistent with the results from ROAM in [14]. The corresponding increase with respect to the independent cascade model could at least partly be explained by how influence was measured – simply counting the average number of activated nodes after some number of iterations is a crude measure. Perhaps more accurate results could be obtained by considering how *likely* it is for a given node to become activated given an initial seed node, that is, its global activation probability. The influence of the target node could then be some linear combination of these probabilities – e.g. their sum, or a weighted sum if one wishes to capture different

strengths of the edges as well.

Finally, the Shapley values for the game described in Section 4.2 decreased very little as ROAM and ROAMeig were run several consecutive times. This is expected since the Shapley values are designed to capture the influence of each node, and for our target nodes we saw how that increased with both heuristics. Thus, as a centrality measure, the Shapley value stands robust against attempts to evade social network analysis by using ROAM and ROAMeig. It is worth noting that in this thesis a rather simple assumption is made on the sphere of influence of a coalition (as used in the cooperative game), namely that it consists of the member of the coalition itself as well as any nodes reachable by at most one hop from a coalition member node. This model could be developed further by e.g. considering only the nodes in the coalition and those which are reachable in at least k different ways from member nodes, which arguably is a more sophisticated assumption.

The main conclusions to be drawn from the experiments are the following. Firstly, the ROAMeig heuristic seems to be at least as good as ROAM, and outperforms it with respect to eigenvector centrality – which it was designed for. Secondly, both algorithms increased the influence of the target node when considering the crude measurement of counting the number of successfully activated nodes. Finally, both heuristics are easily countered by considering the Shapley value for the game described in the thesis, and therefore we propose that the tools and methods from cooperative game theory deserves more attention when it comes to unveil covert networks which we assume are actively trying to conceal themselves by using heuristics similar to ROAM and ROAMeig.

7

Further research

The questions asked in this thesis open up several possible explorations not covered here. What follows are some suggestions for further consideration.

- Is it possible to achieve similar results if we instead of individuals are trying to hide/find entire communities? Of course a more sophisticated heuristic has to be developed. In Waniek et al [14] an algorithm, DICE, is suggested for this purpose. Perhaps this idea could be developed further, e.g. by considering game theoretic centrality measures as was done in this thesis.
- There are other more direct ways to modify eigenvectors during perturbations of the network. Could an algorithm based on these methods be developed with even better performance (in terms of complexity as well as achieved effect)?
- One could consider other spheres of influence – e.g. only neighbours that are reachable in k different ways. These might more realistically reflect influence dynamics in a social network.
- How well do the heuristics examined in this thesis perform if weighted networks are considered (where the weights can represent e.g. trust or strength of friendship)? The influence metrics would of course have to bee modified in order to capture this extra structure.
- A better method of assigning influence score via the influence models could perhaps reveal more realistically how the edge modifications affects how information and ideas flow from the target node and outwards in the network. (In particular one expects that the influence modelled by independent cascade should decrease with each iteration of ROAM or ROAMeig.)

Appendix

A

Supplementary theory

A.1 Calculating the Shapley values for the game

We want to calculate the Shapley value of all players for the game discussed in Section 4.2. To find the value of a node v_i, we consider all possible permutations of the nodes in which v_i would make a positive contribution to the coalition of nodes occurring before itself. Let the set of nodes occurring before node v_i in a random permutation of nodes be denoted by C_i. As before we let $N(v_i)$ and $d(v_i)$ denote the neighbourhood and degree of v_i, respectively.

The key question to ask is: What is the necessary and sufficient condition for node v_i to marginally contribute node $v_j \in N(v_i) \cup \{v_i\}$ to fringe(C_i)? This clearly happens if and only if neither of v_j nor any of *its* neighbours are present in C_i, which formally translates to

$$\Big(N(v_j) \cup \{v_j\}\Big) \cap C_i = \emptyset.$$

We will now prove that this condition holds with probability $\frac{1}{1+d(v_j)}$. The following proposition with proof is taken from [8].

Proposition A.1. *The probability that in a random permutation none of the vertices from $N(v_j) \cup \{v_j\}$ occurs before v_i, where v_j and v_i are neighbours, is $\frac{1}{1+d(v_j)}$.*

Proof: We need to count the number of permutations π that satisfy

$$\pi(v_i) < \pi(v) \;\forall\; v \in \Big(N(v_j) \cup \{v_j\}\Big), \tag{A.1}$$

i.e. permutations in which v_i is mapped to the first position among the vertices in the considered set. To this end, choose $\Big|N(v_j) \cup \{v_j\}\Big|$ positions in the sequence

of elements from $V(G)$. This can be done in $\binom{|V|}{1+d(v_j)}$ ways. In the last $d(v_j)$ chosen positions, place all elements from $\Big(N(v_j) \cup \{v_j\}\Big) \setminus \{v_i\}$. Directly before these, place the element v_i. The number of such line-ups is $(d(v_j))!$, and the remaining elements can now be arranged in $\Big(|V| - \big(1 + d(v_j)\big)\Big)!$ different ways.

Altogether, the number of permutations satisfying condition A.1 is

$$\binom{|V|}{1 + d(v_j)}(d(v_j))!\Big(|V| - \big(1 + d(v_j)\big)\Big)! = \frac{|V|!}{1 + d(v_j)},$$

implying that the probability of randomly choosing one such permutation is

$$\frac{1}{|V|!}\frac{|V|!}{1 + d(v_j)} = \frac{1}{1 + d(v_j)}.$$

□

A.2 The Barabási-Albert (BA) model

This algorithm is used for constructing random scale-free[1] networks. Among many other applications, BA networks are thought to be good approximations of real world social networks because of its scale-freeness. The BA networks are constructed as follows. An initial connected network of m_0 nodes is given. In each iteration, a new node is added and connected to $m \leq m_0$ existing nodes. The probability p_i that the new node will be connected to node i is

$$p_i = \frac{d(i)}{\sum_j d(j)},$$

where $d(i)$ is the degree of node i, and the sum is taken over all pre-existing nodes. Thus it is more likely that the new node gets connected to a node with a high degree rather than one with a low degree. This results in a network where there are a few nodes with a large number of neighbours, while most nodes will have few neighbours.

[1] That is, the networks' edge distribution follows, at least asymtpotically, a power law.

B

Source code

ROAM.py

The original ROAM heuristic.

```
from snap import *
from sys import argv
import operator

def ROAM(*args):
        filename = argv[1]
        budget = int(argv[2])

        print "Friendly reminder: Run all networks through
            FileConverter.py first."

        G = LoadEdgeList(PUNGraph, filename, 0, 1)
        maxCentr = float('-inf')

        #calculate degree centrality
        #NI = node iterator, constructed through the call
        #to the method G.Nodes()

        for NI in G.Nodes():
                degCentr = GetDegreeCentr(G, NI.GetId())*(G
                    .GetNodes()-1)
                if degCentr > maxCentr:
                        targetNode =  NI.GetId()
```

```
22                          maxCentr = degCentr
23
24          if len(argv) == 4:
25                  targetNode = int(argv[3])
26
27          #print old centralities
28          degCentr = GetDegreeCentr(G, targetNode)*(G.
                GetNodes()-1)
29          closeCentr = GetClosenessCentr(G, targetNode)
30          Nodes = TIntFltH()
31          Edges = TIntPrFltH()
32          GetBetweennessCentr(G, Nodes, Edges, 1.0)
33          inBtwCentr = Nodes[targetNode]
34
35          NIdEigenH = TIntFltH()
36          GetEigenVectorCentr(G, NIdEigenH)
37          eigCentr = NIdEigenH[targetNode]
38
39          print "Target node was: " + str(targetNode)+"\n"
40          print('{}{}'.format("Old degree centrality: ",
                degCentr))
41          print('{}{}'.format("Old closeness centrality: ",
                closeCentr))
42          print('{}{}'.format("Old in-betweeness centrality:
                ", inBtwCentr))
43          print('{}{}'.format("Old eigenvector centrality: ",
                 eigCentr)+"\n")
44
45          #find targetNode's friends
46          friends = TIntV()
47          for N in G.GetNI(targetNode).GetOutEdges():
48                  friends.Add(N)
49
50          # Get subgraph induced by the neighbours of
                targetNode.
51          subGraph = GetSubGraph(G, friends)
52
53          degCentrVec = {}
54
55          for NI in subGraph.Nodes():
56                  degCentr = GetDegreeCentr(subGraph, NI.
                        GetId())*(G.GetNodes()-1)
57                  degCentrVec[NI.GetId()] = degCentr
58
59          #locate the b-1 friends with highest centrality
```

```
60          sortedCntrVec = sorted(degCentrVec.items(), key=
                operator.itemgetter(1))
61          if len(sortedCntrVec) <= budget:
62                  del sortedCntrVec[2:len(sortedCntrVec)-(
                        budget-1)]
63
64          #let the friend with highest centrality be denoted
                by v0
65          v0 = sortedCntrVec[0]
66
67          #remove edge between targetNode and v0
68          G.DelEdge(targetNode, v0[0])
69
70          #add edges between v0 and the b-2 other friends
71          for key, value in sortedCntrVec:
72                  if key != v0[0]:
73                          G.AddEdge(key,v0[0])
74
75          #save graph
76          SaveEdgeList(G, "output.txt", "Save as tab-
                separated list of edges");
77
78          #print new centralities
79          degCentr = GetDegreeCentr(G, targetNode)*(G.
                GetNodes()-1)
80          closeCentr = GetClosenessCentr(G, targetNode)
81          Nodes = TIntFltH()
82          Edges = TIntPrFltH()
83          GetBetweennessCentr(G, Nodes, Edges, 1.0)
84          inBtwCentr = Nodes[targetNode]
85
86          NIdEigenH = TIntFltH()
87          GetEigenVectorCentr(G, NIdEigenH)
88          eigCentr = NIdEigenH[targetNode]
89
90          print('{}{}'.format("New degree centrality: ",
                degCentr))
91          print('{}{}'.format("New closeness centrality: ",
                closeCentr))
92          print('{}{}'.format("New in-betweeness centrality:
                ", inBtwCentr))
93          print('{}{}'.format("New eigenvector centrality: ",
                 eigCentr))
94
95  if __name__ == '__main__':
96      ROAM(argv)
```

ROAMeig.py

The modified ROAM heuristic.

```
from snap import *
from sys import argv
import numpy
import operator
import csv

def ROAMeig(*args):
        filename = argv[1]
        budget = int(argv[2])

        print "Friendly reminder: Run all networks through
            FileConverter.py first."
        G = LoadEdgeList(PUNGraph, filename, 0, 1)
        maxCentr = float('-inf')

        #create adjacency list
        with open(filename, 'rb') as f:
                reader = csv.reader(f,delimiter='\t')
                for i in range(0, 4):
                        reader.next()
                adj_list = list(reader)
        f.closed

        #matrix size
        flatten = map(int, [val for sublist in adj_list for
             val in sublist])
        n = max(flatten)
        A = numpy.zeros((n,n))

        #build matrix
        for i in adj_list:
                A[int(i[0])-1][int(i[1])-1] = 1
                A[int(i[1])-1][int(i[0])-1] = 1

        #calculate square matrix
        A2 = numpy.linalg.matrix_power(A, 2)

        #matrix sums
        sum = numpy.identity(n) + A + A2

        #calculate degree centrality
        #NI = node iterator, constructed through the call
        #to the method G.Nodes()
```

```
42
43          degCentrVec = numpy.zeros(n)
44
45          for NI in G.Nodes():
46                  degCentr = GetDegreeCentr(G, NI.GetId())*(G
                        .GetNodes()-1)
47                  degCentrVec[int(NI.GetId())-1] = degCentr
48                  if degCentr > maxCentr:
49                          targetNode =  NI.GetId()
50                          maxCentr = degCentr
51
52          if len(argv) == 4:
53                  targetNode = int(argv[3])
54
55          #print old centralities
56          degCentr = GetDegreeCentr(G, targetNode)*(G.
                GetNodes()-1)
57          closeCentr = GetClosenessCentr(G, targetNode)
58          Nodes = TIntFltH()
59          Edges = TIntPrFltH()
60          GetBetweennessCentr(G, Nodes, Edges, 1.0)
61          inBtwCentr = Nodes[targetNode]
62
63          NIdEigenH = TIntFltH()
64          GetEigenVectorCentr(G, NIdEigenH)
65          eigCentr = NIdEigenH[targetNode]
66
67          print "Target node was: " + str(targetNode)+"\n"
68          print('{}{}'.format("Old degree centrality: ",
                degCentr))
69          print('{}{}'.format("Old closeness centrality: ",
                closeCentr))
70          print('{}{}'.format("Old in-betweeness centrality:
                ", inBtwCentr))
71          print('{}{}'.format("Old eigenvector centrality: ",
                 eigCentr)+"\n")
72
73          #find targetNode's friends
74          friends = TIntV()
75          for N in G.GetNI(targetNode).GetOutEdges():
76                  friends.Add(N)
77
78          # Get subgraph induced by the neighbours of
                targetNode.
79          subGraph = GetSubGraph(G, friends)
80
```

```
81          degMassCentrVec = {}
82
83          for NI in subGraph.Nodes():
84                  targetRow = sum[int(NI.GetId())-1]
85                  degMassCentr = numpy.dot(targetRow,
                        degCentrVec)
86                  degMassCentrVec[NI.GetId()] = degMassCentr
87
88          #locate the b-1 friends with highest centrality
89          sortedMassCntrVec = sorted(degMassCentrVec.items(),
                key=operator.itemgetter(1))
90          if len(sortedMassCntrVec) <= budget:
91                  del sortedMassCntrVec[2:len(sortedMassCntr)
                        -(budget-1)]
92
93          #let the friend with highest centrality be denoted
                by v0
94          v0 = sortedMassCntrVec[0]
95
96          #remove edge between targetNode and v0
97          G.DelEdge(targetNode, v0[0])
98
99          #add edges between v0 and the b-2 other friends
100         for key, value in sortedMassCntrVec:
101                 if key != v0[0]:
102                         G.AddEdge(key,v0[0])
103
104         #save graph
105         SaveEdgeList(G, "outputeig.txt", "Save as tab-
                separated list of edges");
106
107         #print new centralities
108         degCentr = GetDegreeCentr(G, targetNode)*(G.
                GetNodes()-1)
109         closeCentr = GetClosenessCentr(G, targetNode)
110         Nodes = TIntFltH()
111         Edges = TIntPrFltH()
112         GetBetweennessCentr(G, Nodes, Edges, 1.0)
113         inBtwCentr = Nodes[targetNode]
114
115         NIdEigenH = TIntFltH()
116         GetEigenVectorCentr(G, NIdEigenH)
117         eigCentr = NIdEigenH[targetNode]
118
119         print('{}{}'.format("New degree centrality: ",
                degCentr))
```

```
120         print('{}{}'.format("New closeness centrality: ",
                closeCentr))
121         print('{}{}'.format("New in-betweeness centrality:
                ", inBtwCentr))
122         print('{}{}'.format("New eigenvector centrality: ",
                 eigCentr))
123
124 if __name__ == '__main__':
125     ROAMeig(argv)
```

indepCascade.py

The independent cascade influence model.

```
from snap import *
from sys import argv
from collections import deque
import numpy
import operator
import csv

def IndependentCascade(filename,targetNode):
        G = LoadEdgeList(PUNGraph, filename, 0, 1)
        nrOfEdges = G.GetEdges()
        uniformProb = 0.4
        activeSet = deque([targetNode])
        activatedNodes = set([targetNode])
        while activeSet:
                temp = activeSet.popleft()
                for N in G.GetNI(temp).GetOutEdges():
                        if N not in activatedNodes and
                            numpy.random.uniform() <=
                            uniformProb:
                                activeSet.append(N)
                                activatedNodes.add(N)
        with open('indepCascade.txt', 'w') as f:
                f.write(str(activatedNodes))
        f.closed
        print "Number of activated nodes: "+str(len(
            activatedNodes))
if __name__ == '__main__':
    IndependentCascade(argv[1],int(argv[2]))
```

linearThresh.py

The linear threshold influence model.

```
from snap import *
from sys import argv
from collections import deque
from random import randint
import numpy
import operator
import csv

def LinearThreshold(filename,targetNode):
        G = LoadEdgeList(PUNGraph, filename, 0, 1)
        nrOfEdges = G.GetEdges()
        activeSet = deque([targetNode])
        prevActiveSet = set([])
        activatedNodes = set([targetNode])
        thresholds = {}
        for NI in G.Nodes():
                nodeId = NI.GetId()
                degree = int(GetDegreeCentr(G, nodeId)*(G.
                    GetNodes()-1))
                thresholds[nodeId] = randint(0,degree)
        while activeSet:
                temp = activeSet.popleft()
                prevActiveSet.add(temp)
                for NI in G.Nodes():
                        intersect = prevActiveSet.
                            intersection(NI.GetOutEdges())
                        nodeId = NI.GetId()
                        if nodeId not in activatedNodes and
                             len(intersect) >= thresholds[
                            nodeId]:
                                activeSet.append(nodeId)
                                activatedNodes.add(nodeId)
                                prevActiveSet.add(nodeId)
                prevActiveSet.remove(temp)
        with open('linearThreshold.txt', 'w') as f:
                f.write(str(activatedNodes))
        f.closed
        print "Number of activated nodes: "+str(len(
            activatedNodes))
if __name__ == '__main__':
    LinearThreshold(argv[1],int(argv[2]))
```

FileConverter.py

Converts adjacency lists (from e.g. SNAP) to tab separated format.

```
from snap import *
from sys import argv

def FileConverter(filename):

        #load graph
        G = LoadEdgeList(PUNGraph, filename, 0, 1)

        #save graph
        SaveEdgeList(G, "formatted_network.txt", "Save as
            tab-separated list of edges");

if __name__ == '__main__':
    FileConverter(argv[1])
```

shapleyg1.py

Calculates shapley values of each node in a given graph for the game g_1 considered in the thesis.

```
from snap import *
from sys import argv
import numpy
import operator
import csv

def ShapleyValue(filename,targetNode):
        print "Ouput written to shapley.txt."
        G = LoadEdgeList(PUNGraph, filename, 0, 1)

        SV = [None]*G.GetNodes()
        for NI in G.Nodes():
                nodeId = NI.GetId()
                SV.insert(nodeId-1,1/(1+(GetDegreeCentr(G,
                    nodeId)*(G.GetNodes()-1))))
                for N in NI.GetOutEdges():
                        update = SV[nodeId-1] + 1/(1+(
                            GetDegreeCentr(G, N)*(G.
                            GetNodes()-1)))
                        SV.insert(nodeId-1,update)
        with open('shapley.txt', 'w') as f:
                for item in SV:
```

```
                            f.write("%s\n" % item)
        f.closed
        print "Node "+str(targetNode)+" has Shapley value:
            "+str(SV[int(targetNode)-1])
if __name__ == '__main__':
    ShapleyValue(argv[1],argv[2])
```

scaleFreeGen.py

Generates a scale free network.

```
import snap
from sys import argv

def ScaleFreeGen(nrNodes, deg):
        Rnd = snap.TRnd()
        UGraph = snap.GenPrefAttach(nrNodes, deg, Rnd)

        #save graph
        snap.SaveEdgeList(UGraph, "scale_free.txt", "Save
            as tab-separated list of edges");

if __name__ == '__main__':
    ScaleFreeGen(int(argv[1]), int(argv[2]))
```

Bibliography

[1] J. Bondy and U.S.R. Murty. *Graph Theory*. Springer, Berlin, 2008. Cited on pages 3 and 8.

[2] J. Granholm. Some cyclic properties of graphs with local Ore-type conditions. LiTH-MAT-EX–2016/04–SE, Linköping University, Sweden, 2016. Cited on page 3.

[3] M. Kearns, A. Roth, Z. S. Wu, and G. Yaroslavtsev. Private algorithms for the protected in social network search. In *Proceedings of the National Academy of Sciences*, volume 113, pages 913–918, 2016. Cited on page 1.

[4] V. Krebs. Mapping networks of terrorist cells. *CONNECTIONS*, 24(3):43–52, 2002. Cited on page 21.

[5] J. I. Lane, V. Stodden, S. Bender, and H. Nissenbaum, editors. *Privacy, big data, and the public good: frameworks for engagement*. Cambridge University Press, Cambridge, 2014. Cited on page 1.

[6] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014. Cited on page 21.

[7] C. Li, O. Li, P. van Mieghem, H. E. Stanley, and H. Wang. Correlation between centrality metrics and their application to the opinion model. *European Physical Journal B*, 88(65), 2015. Cited on page 8.

[8] T. Michalak, K. V. Aadithya, P. L. Szczepanski, B. Ravindran, and N. R. Jennings. Efficient computation of the shapley value for game-theoretic network centrality. *J. Artif. Int. Res.*, 46(1):607–650, January 2013. ISSN 1076-9757. URL `http://dl.acm.org/citation.cfm?id=2512538.2512553`. Cited on page 37.

[9] T. Michalak, T. Rahwan, and M. Wooldridge. Strategic social network analysis, 2017. URL `https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14941`. Cited on page 1.

[10] R. B. Myerson. Graphs and cooperation in games. *Mathematics of Operations Research*, 2(3):225–229, 1977. Cited on pages 18 and 19.

[11] N. Perra and S. Fortunato. Spectral centrality measures in complex networks. *Phys. Rev. E*, 78:036107, Sep 2008. doi: 10.1103/PhysRevE.78.036107. URL https://link.aps.org/doi/10.1103/PhysRevE.78.036107. Cited on page 9.

[12] L. S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953. Cited on page 17.

[13] N. R. Suri and Y. Narahari. Determining the top-k nodes in social networks using the shapley value. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '08, pages 1509–1512, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-2-3. URL http://dl.acm.org/citation.cfm?id=1402821.1402911. Cited on page 20.

[14] M. Waniek, T. Michalak, T. Rahwan, and M. Wooldridge. Hiding Individuals and Communities in a Social Network. *ArXiv e-prints*, August 2016. Cited on pages 1, 13, 22, 31, and 33.

Index